Synthesis of SI Circuits from Burst-Mode Specifications

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

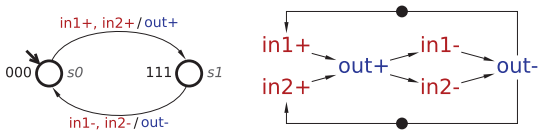
***Abstract*—In this paper, we present a new workflow that is based on the conversion of Extended Burst-Mode (XBM) specifications to Signal Transition Graphs (STGs). While XBMs offer a simple design entry to specify asynchronous circuits, they cannot be synthesised into speed-independent (SI) circuits, due to the 'burst mode' timing assumption inherent in the model. Furthermore, XBM synthesis tools are no longer supported, and there are no dedicated tools for formal verification of XBMs. Our approach addresses these issues, by granting the XBMs access to sophisticated synthesis and verification tools available for STGs, as well as the possibility to synthesise SI circuits. Experimental results show that our translation only linearly increases the model size and that our workflow achieves a much improved synthesis success rate, with a 33% average reduction in the literal count.**

I. Introduction

Asynchronous circuits remove the use of a global clock signal in favour of local synchronisation between its components [1]. They provide many advantages including higher performance, enhanced robustness and lower power consumption [2], as well as removal of clock-related issues such as clock skew [3]. Static timing analysis methods [4] are essential for measuring the time performance of circuits, by examining the delays and transition times of their components. Paradigms like WAITX elements [5] fall under an important class of SI asynchronous circuits, where, following the classical Muller's approach [6], each gate is regarded as an atomic evaluator of a Boolean function with a delay associated to the output. In the SI framework, this delay is positive and finite, but also variable and unbounded. As such, the circuit must work correctly regardless of its gates' delays, and the wires are assumed to have negligible delays.

Burst-mode (BM) specification [7], [8] is an established Finite State Machine introduced as a simple design entry for asynchronous circuit design. BMs allow signals to change in groups called bursts, where they may arrive in any order and time. Each burst is connected between two states and contains a non-empty set of inputs that precede a finite set of outputs. All states must be entered with the same set of inputs to simplify hazard-free synthesis (unique entry condition), and no input burst can be a subset of another burst from the same incoming state to prevent non-deterministic transitions (maximal set property). However, these inputs cannot change concurrently with outputs and the BM's synthesis tool, MINIMALIST [8], cannot run on a modern-day operating system (OS) due to 32-bit binaries that require an older OS. Also, there is no tool that supports formal verification of BMs.

XBM specification [9] addresses the input-output concurrency issue of BMs by expanding it with conditionals and "don't cares". Conditionals are level-sensitive inputs that determine control flow based on signal levels, while "don't cares" are inputs that can change concurrently with outputs. Moreover, input bursts must have unique conditional values or are not a subset of another burst from the same incoming state, including "don't cares" transitions (distinguishability constraint). Despite this improvement, the XBM's synthesis tool, 3D [9], is no longer available for download and there is also no tool that supports formal verification of XBMs. Figure 1(a) illustrates the C-element specification using the BM and XBM formalisms, where notations of conditionals and "don't cares" can be found in Figure 4(a) and Figure 5(a) respectively.



(a) BM and XBM model (b) STG model

Fig. 1. Specification of C-element

Alternatively, STGs [10], [11] are circuit-based Petri Nets that model the rising and falling edges of signals. STGs consist of a set of places that may contain a token, and a set of transitions associated with the rising and falling edges of signals (input, output or internal). Arcs from places to transitions and transitions to places denote the flow of tokens, where a transition is enabled if all preceding places have tokens. When an enabled transition is fired, it consumes tokens from all preceding places and produces tokens to all succeeding places. One of the key advantages for STGs is their wide support from many established synthesis and verification tools, such as PETRIFY [12] and MPSAT [13]. Figure 1(b) also illustrates the C-element specification using the STG formalism.

In this paper, we propose a new workflow based on the conversion of XBMs to STGs, to address the issues of unavailable synthesis and verification tools for BMs and XBMs. Unlike previous approaches [14]–[18], our approach preserves the properties of XBMs and STGs, and demonstrates improvements to literal count as shown in Section IV. [14] enables specifications of mixed-mode asynchronous/synchronous systems and arbitration with its modelling primitives, however its level transitions break the STG's consistency property. [15], [16] enables specifications of heterogeneous systems with implicit input and output bursts, but they resolve complete state coding (CSC) conflicts using a different model. [17] synthesises more robust XBMs into quasi-delay insensitive circuits to enable CMOS-UDSM technology, although no verification of the XBMs were included. [18] shows how an STG specification can be used for designing, before choosing PETRIFY or 3D once it is partitioned into several XBMs. Nevertheless, the STG has missing XBM

978-3-9819263-5-4/DATE21/©2021 EDAA

behaviour due to the assumptions made at the partition step. Also, no benchmarks were found and no extensions for a verification flow nor XBM to STG conversion were covered.

For our approach, it will be incorporated into the WORKCRAFT toolkit [19]. WORKCRAFT provides rich support for STGs, including a graphical front-end for editing and simulation, and convenient mechanisms for verification and synthesis of constructed STGs via PETRIFY and MPSAT back-ends. Furthermore, WORKCRAFT has a plugin-based architecture, allowing new plugins and new back-ends to be easily integrated. Thus, we implement a plugin that supports the design automation of XBMs, granting it access to the established synthesis and verification tools via STG conversion. What separates our approach from [14]–[18] is the inclusion of a verification flow, allowing verification of BMs and XBMs to be performed at the design level. The contribution of our paper is as follows: (i) novel workflow for synthesis and verification of XBM specifications (Section II); (ii) method for conversion from XBMs to STGs (Section III); (iii) evaluation of proposed workflow and its design automation on a number of benchmarks (Section IV).

II. PROPOSED WORKFLOW

Figure 2 shows our proposed workflow describing the XBM to STG conversion flow in WORKCRAFT, from model design to conformation of the synthesised circuit. Both design and verification flows are also included using WORKCRAFT's front-end and back-ends respectively. Below, we illustrate our workflow using an example (where each step is annotated on Figure 2):

1) The user designs or imports their BM/XBM using WORKCRAFT. They may then validate the behaviour of the BM/XBM, using interactive simulation and observing its traces.
2) The BM/XBM can now be verified against the described BM/XBM properties, including the maximal set property and distinguishability constraint. Verification results are then returned to the user as a report, indicating a pass or highlighting the violations found for debugging purposes.
3) Conversion of the BM/XBM to STG may now be performed, where states, bursts, conditionals and "don't cares" are translated to a set of places, explicit bursts, elementary cycles and "don't care" transitions respectively. Details of these conversions are covered in Section III.
4) Once an STG is obtained, it can be verified against the implementability properties for STGs. This include checks for consistency, deadlock freeness, output persistency, input properness and detection of CSC conflicts. Again, verification results are returned to the user as a report, indicating a pass or highlighting the violations found.
5) The user may now synthesise their STG into an SI circuit as a complex gate, generalised C-element or standard C-element using either PETRIFY or MPSAT.
6) Lastly, conformation to the original XBM and/or translated STG can be performed on the synthesised circuit. This ensures the circuit is correctly implemented, as synthesis tools can be complex and may introduce bugs. Additionally, manual modification is error prone and must be verified accordingly.

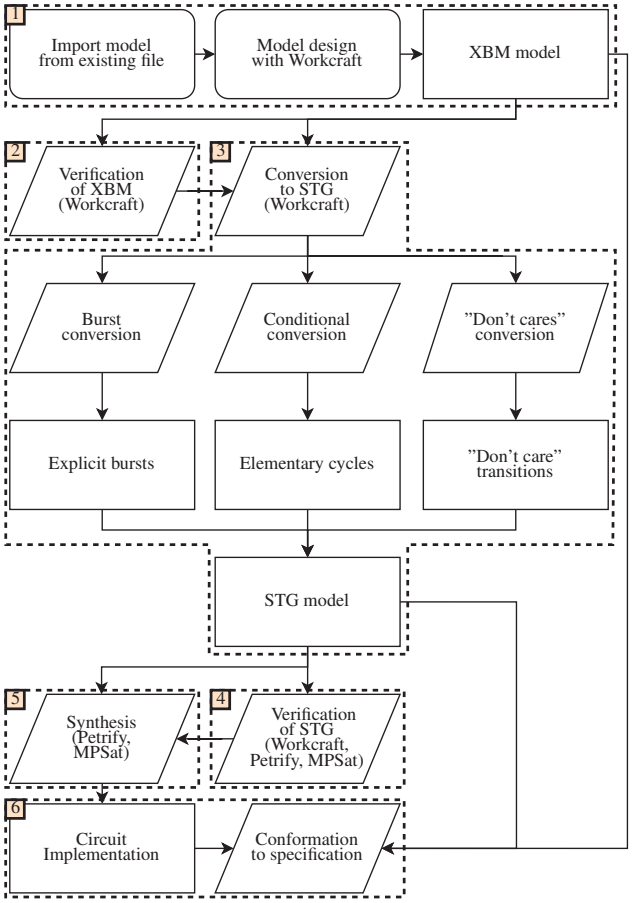
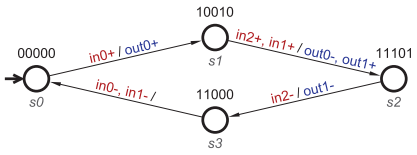


Fig. 2. Proposed workflow based on WORKCRAFT

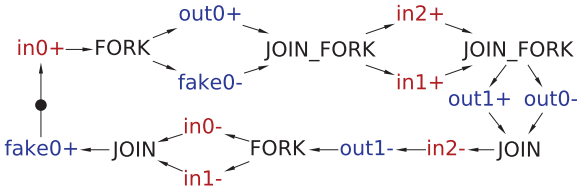
III. CONVERSION FROM XBM TO STG

In our methodology, we convert XBM components (states, bursts, conditionals and "don't cares") to their STG counterparts (set of places, explicit bursts, elementary cycles and "don't care" transitions). The simplest conversion are states, where each state is translated into a set of places and these places are assigned a token, if the given state is an initial state. This ensures that the STG is initially 1-safe and not deadlocked. Conversion of other XBM components are more complex and require several considerations to ensure the XBM properties are preserved in the translated STG, such as bursts without outputs and conditional setup and hold times.

1) Bursts: Bursts are a group of signals that may change in any order and time, where each burst has a non-empty set of inputs that precede a finite set of outputs. In [8] and [9], inputs are allowed to arrive in any order, indicating they can be accepted concurrently. However, this was not specified for outputs, suggesting they can be accepted in any arbitrary order (whether sequentially or concurrently). In our work, we consider outputs are accepted concurrently. One reason is this follows the forementioned definition of bursts, which does not enforce any ordering to the signals. Another reason is post-



(a) Empty output bursts in XBM



(b) 'Fake' output transitions in STG

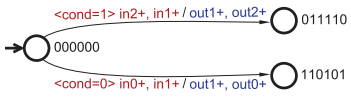
Fig. 3. XBM to STG conversion of bursts without outputs

processing methods like concurrency reduction can be used to optimise the ordering of outputs after translation. We highlight this optimisation as part of our future work.

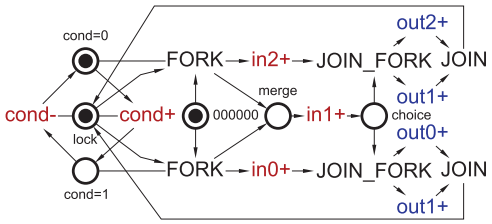
For some bursts, they may have an empty set of outputs. [8] defines that a "dummy output" is created for all bursts without outputs to signify an internal change. In our conversion, we create a fake output transition in the resultant STG for every burst without any outputs in the XBM. To ensure these fake outputs remain consistent, their opposite-edged transition is inserted after an opposite-edged transition for one of their corresponding inputs has occurred. As an example, consider burst 'i1−, i0− /' in Figure 3: the falling transition for the fake output is inserted after the rising transition of input i0.

2) Conditionals: Conditionals are level-sensitive inputs that determine control flow based on signal values. They may continuously flip between state 0 and state 1 until they are sampled, once a corresponding input terminates with a rising or falling transition (setup time). The conditional must then remain stable until all corresponding outputs occur (hold time).

In our work, we translate conditionals to elementary cycles, due to their functional similarities. Elementary cycles for signals are a pair of places and a set of rising and falling transitions, where places represent when the signal is 0 or 1. For the elementary cycles to remain stable, a number of lock places and replicated pairs of rising and falling transitions must be generated. This value is based on the conjoined set of inputs, which terminate with a rising or falling transition from each burst that share the same incoming state. The purpose of these locks is to prevent the elementary cycle from repeatedly firing its set of rising (falling) transitions, when one of its connected input transitions has fired (satisfying the setup time). Once all connected output transitions have occurred, locks regain their tokens and re-enable the elementary cycle for firing (satisfying the hold time). Figure 4 shows the conversion of conditionals to elementary cycles, where each lock place is connected to every pair of rising and falling transitions via a read arc. Additionally, each lock place is also connected to a set of inputs and a set of outputs with consuming and producing arcs respectively.

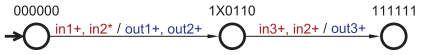


(a) Conditional in XBMs

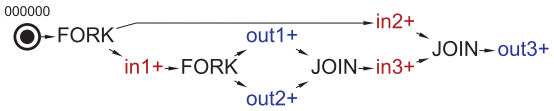


(b) Elementary cycle in STGs

Fig. 4. XBM to STG conversion of conditionals



(a) The "don't care" in2 in XBM



(b) The forking and joining of "don't care" in2 in STG

Fig. 5. XBM to STG conversion of "Don't cares"

3) "Don't cares": "Don't cares" are inputs that may continuously appear in multiple bursts, until they terminate with either a rising or falling transition. Termination of the "don't care" is dependant on its original encoded value, where it terminates with a rising (falling) transition if it was state 0 (1) before appearing as a "don't care". Moreover, as defined in [9], "don't care" changes are monotonic meaning it may only change once during its appearance in multiple bursts.

Like [18], our work connects the "don't care" from the first burst it appears in with a producing arc, before it is connected to the next burst it terminates in with a consuming arc. This is shown as an example in Figure 5, where the "don't care" (in2∗) is connected from the burst (in1+, in2∗ / out1+, out2+) via a forked producing arc, and to the burst (in3+, in2+ / out3+) via a joint consuming arc. To simplify the condition where a "don't care" terminates in multiple paths, we generate a new common place that the "don't care" is connected to with a producing arc. This place is then connected to all bursts that the "don't care" terminates in, acting as a common enabler for each corresponding burst. If a "don't care" appears in a loop then we assume it must terminate before exiting the loop.

IV. Experimental Results and Analysis

The benefits of our workflow are demonstrated by converting published XBMs to STGs using our methodology, where Table I shows our experimental results. Columns 'XBM size' and 'STG size' shows the comparison between the sizes of the XBM and STG designs for each specification. XBM size represents the total number of states, bursts (i.e. arcs) and

labelled signals of each burst, while STG size represents the total number of explicit places, signal transitions and arcs (where transition-transition arcs that have implicit places are counted as one arc). These results show our XBM to STG translation only linearly increases the size of the specification by an average factor of 2–3, avoiding the state explosion problem [20]. This is also particularly beneficial for ensuring the translated specification remains comprehensive, especially when synthesising them into circuits at a smaller complexity as demonstrated in [21]. The other column 'Literal count / overhead vs best solution' shows the comparison between the number of literals used in the synthesised circuits of MINIMALIST, 3D, PETRIFY and MPSAT. Each literal count with a percentage highlights their overhead over the smallest literal count of that row. The second final row shows the overall success rate for each tool that attempted to synthesis each specification, where all '–' results are considered a failed attempt. The final row shows the overall average overhead of each tool over the best solution (i.e. smallest literal count), based on a 100% success rate ratio. These results show our approach has an improved success rate of synthesising the specifications, while also having an overall smaller literal count. This is reflected by the high success rates and lower literal counts from PETRIFY and MPSAT over MINIMALIST and 3D.

TABLE I
COMPARISON BETWEEN BM/XBM AND STG FORMALISMS AND TOOLS

Specification	XBM size	STG size	Literal count / Overhead vs best solution			
			MINIMALIST	3D	PETRIFY	MPSAT
ack-xbm-si	38	91	–	27/ 17%	26/13%	**23**
biu-dma2fifo	44	110	–	33/ 65%	24/20%	**20**
biu-fifo2dma	33	91	–	–	**23**	25/ 9%
bundled-data-2p	33	79	–	–	23/ 5%	**22**
counter-6	18	46	32/256%	**9**	13/44%	13/44%
celement	10	22	12/140%	**5**	**5**	**5**
concur-mixer	27	55	37/185%	**13**	15/15%	15/15%
diffeq-alu1	50	132	100/223%	38/ 23%	**31**	37/19%
diffeq-alu2	99	132	–	**83**	90/ 8%	102/23%
diffeq-mul1	27	66	–	31/ 48%	**21**	22/ 5%
diffeq-mul2	19	46	–	16/ 23%	**13**	**13**
dme	38	50	28/100%	19/ 36%	15/ 7%	**14**
dme-fast	45	78	58/287%	26/ 73%	20/33%	**15**
dram-ctrl	75	165	78/136%	39/ 18%	38/15%	**33**
fifocellctrl	15	26	–	11/ 22%	11/22%	**9**
gcd-controller	92	336	–	120/ 41%	–	**85**
hp-ir	33	61	13/ 63%	9/ 13%	**8**	**8**
hp-ir-it-ctrl	59	140	78/100%	**39**	**39**	46/18%
hp-ir-rf-ctrl	56	92	54/ 69%	–	–	**32**
imec-alloc-outb	35	41	31/ 94%	22/ 38%	**16**	17/ 6%
imec-sbuf-ramw	34	64	–	28/ 8%	**26**	30/15%
martin-qelement	16	16	13/ 86%	10/ 43%	**7**	**7**
nowick-basic	21	44	14/ 75%	14/ 75%	9/13%	**8**
token-distributor	48	48	44/ 57%	36/ 29%	**28**	**28**
pe-send-ifc	70	166	98/123%	**44**	49/11%	49/11%
sbuf-send-ctrl	35	83	40/ 60%	30/ 20%	26/ 4%	**25**
sbuf-send-pkt2	21	57	–	18/ 50%	13/ 8%	**12**
scsi-isend	57	125	99/254%	46/ 64%	39/39%	**28**
scsi-isend-p	53	123	77/133%	57/ 73%	40/21%	**33**
scsi-trcv-bm	57	127	101/216%	43/ 34%	37/16%	**32**
scsi-tsend-bm	58	112	74/174%	48/ 78%	43/59%	**27**
select2p	28	85	–	24/ 71%	**14**	**14**
tangram-mixer	31	35	16/100%	11/ 38%	10 /25%	**8**
two-ticks-if	31	48	10/100%	12/140%	**5**	**5**
Synthesis success rate			65%	91%	94%	100%
Average overhead			213%	50%	21%	17%

V. CONCLUSION

In this paper, we address the issues of unavailable synthesis and verification tools for BMs and XBMs by proposing a new workflow, based on the conversion of XBMs to STGs. Translation of XBM components to their STG counterparts were covered, along with a demonstration of the design, conversion, synthesis and verification steps of our approach given as a use case. Experimental result demonstrate that our conversion only linearly increases the model size and that our workflow achieves a much improved synthesis success rate, with a 33% average reduction in the literal count.

Future Directions: we consider the conversion of distributed XBMs to STGs for monolithic specifications, as well as the optimisation of output signal event ordering to further improve our synthesised circuit results.

Acknowledgment: This work was funded by Dialog Semiconductor via A×A project and Alex Chan's studentship, with partial support from EPSRC EP/N023641/1. We would like to thank Prof. Steve Nowick for his advice on MINIMALIST.

REFERENCES

[1] J. Sparso and S. Furber, *Principles of Async. Circuit Design*, 2002.
[2] C. J. Myers, *Asynchronous Circuit Design*. John Wiley Sons, Inc., 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] N. Xiromeritis *et al.*, "Graph-based STA for asynchronous controllers," in *PATMOS*, 2019, pp. 9–16.
[5] V. Khomenko *et al.*, "WAITX: An arbiter for non-persistent signals," in *ASYNC*, 2017, pp. 33–40.
[6] D. Muller and W. Bartky, "A theory of asynchronous circuits," in *Int. Symp. Theory of Switching*, 1959, pp. 204–243.
[7] S. Nowick, "Automatic synthesis of burst-mode asynchronous controllers," Ph.D. dissertation, Stanford University, 1993.
[8] S. Nowick and D. Dill, "Synthesis of asynchronous state machines using a local clock," in *ICCD*, 1991, pp. 192–197.
[9] K. Yun and D. Dill, "Automatic synthesis of extended burst-mode circuits," *IEEE Trans. CAD*, vol. 18, no. 2, pp. 101–132, 1999.
[10] L. Rosenblum and A. Yakovlev, "Signal graphs: From self-timed to timed ones," in *Int. Workshop on Timed Petri Nets*, 1985, pp. 199–206.
[11] T. Chu, "Synthesis of self-timed VLSI circuits from graph-theoretic specifications," MIT, USA, Tech. Rep., 1987.
[12] J. Cortadella *et al.*, "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 *et al.*, "Logic synthesis for asynchronous circuits based on petri net unfoldings and incremental SAT," in *ACSD*, 2004, pp. 16–25.
[14] P. Vanbekbergen *et al.*, "A generalized signal transition graph model for specification of complex interfaces," in *EDAC*, 1994, pp. 378–384.
[15] F. Mendes *et al.*, "A novel tool for synth. by direct mapping of async. circuits from extended STG specifications," in *VLSID*, 2018, pp. 451–452.
[16] H. Delsoto *et al.*, "A tools flow for synthesis of async. control circuits from extended STG specifications," in *LASCAS*, 2019, pp. 225–228.
[17] D. L. Oliveira *et al.*, "A tools flow for extended burst-mode state machines design with enhanced robustness," in *INTERCON*, 2020, pp. 1–4.
[18] J. Beister, G. Eckstein, and R. Wollowski, "From STG to extended-burst-mode machines," in *ASYNC*, 1999, pp. 145–158.
[19] "Workcraft," https://workcraft.org/, 2006, accessed: 18-09-2020.
[20] A. Valmari, "The state explosion problem," in *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets*, 1996, pp. 429–528.
[21] P. Mattheakis and C. Sotiriou, "Polynomial complexity asynchronous control circuit synthesis of concurrent specifications based on burst-mode FSM decomposition," in *VLSID*, 2013, pp. 251–256.