Two-Dimensional Regular Expressions for Compositional Bus Protocols

Kathi Fisler
WPI Department of Computer Science
Worcester, MA 01609 USA
kfisler@cs.wpi.edu

***Abstract*—Bus and interconnect protocols contain a few core operations (such as read and write transfers) whose behaviors interleave to form complex executions. Specifications of the core operations should be flexible enough that simple composition operators suffice for capturing most interleavings, even in the presence of common hardware issues such as glitches. Oliveira and Hu proposed a form of pipelined regular expressions to specify atomic protocol compositions, but they abstracted away clocking and glitches. This paper uses the AMBA-2 specification to argue that a loosely-synchronized form of regular expressions handles such timing subtleties while retaining the simplicity of Oliveira and Hu's pipelined compositions.**

I. Introduction

Last year, we attempted to formalize portions of the AMBA-2 interface specification for on-chip data transfers [1]. AMBA-2 uses timing diagrams extensively to present its protocols. Having worked on formalizing timing diagrams for many years [2], our goal was to explore whether formal diagrams were suitable for capturing such a complex specification. The experience was highly instructive: while formalized timing diagrams (ours or Amla et al's [3]) could capture the individual diagrams in the specification, they were too concrete to capture the abstract compositional protocols underlying the diagrams. Oliviera and Hu's pipelined regular expressions [4] avoid some the diagrams' shortcomings, but overconstrain timing issues that the diagrams support naturally. This paper describes our early-stage efforts to identify specification language constructs that capture the structure of AMBA-2 while drawing on the best features of both timing diagrams and pipelined regular expressions. As (readable) diagrams are generally more restrictive semantically than text, this exploratory work extends textual regular expressions. Finding a readable diagrammatic notation that supports the constructs we identify is an interesting problem for future work.

II. The Structure of Bus Specifications

Figure 1 reproduces several AMBA-2 diagrams. The first row (document figures 5-9 and 5-11) shows basic read and write transfers, the second (document figure 5-12) shows a burst sequence of write transfers, and the third (document figure 5-13) shows an interleaved sequence of read and write transfers. The latter two diagrams are pipelined compositions of the first two. A good formalization of this specification would explicitly capture the basic transfers and exploit composition operators to derive the complex behaviors. A closer look at the diagrams reveals several issues that a formal rendering must address to achieve this goal:

1) Pipelining is required, as seen in the overlapping instances of the write protocol in diagram 5-12.
2) The number of clock cycles between events must be allowed to vary between the basic and complex diagrams; clock specifications in the basic diagram may indicate alignment with edges rather than cycle counts.
3) The values on signals may differ when a basic protocol appears in composition. The basic write diagram (5-11) ends with PSEL low, but the burst diagram (5-12) preserves the prior high value across adjacent transfers.
4) Names equate values across signals in the diagrams.
5) Glitches and regions of potential signal instability (shown shaded) must be captured and can vary in length.

Other diagrams (not shown here) refine the basic protocols with error handling and other features. The observations in this paper are necessary, but not sufficient, for those cases.

Regular expressions are common for modeling protocols. As Oliveira and Hu argue [4], however, defining an interleaved sequence of protocols as a standard regular expression is problematic because such sequences capture parallel executions which vastly complicate the expressions. Furthermore, such expressions fail to reuse the specifications of the basic protocols. Their elegant proposal was to introduce a variant of the concatenation operator (denoted @ instead of ;) to indicate the point in a regular expression at which a parallel execution could begin. For example, the expression $a;b@c$ allows a new instance of the expression (starting from a) to begin concurrently with c (as the first term after the @ operator). With this notation, they can cleanly specify pipelined compositions of protocols whose steps execute in fixed numbers of cycles.

When the number of cycles between protocol steps is not fixed, however, the regular expressions show a key limitation. Consider the HADDR, HWDATA, and HREADY signals from the basic write protocol (5-11). Associating each character in an expression with a clock cycle, we could capture these as

$$\text{HADDR} = Addr_1\,;\,dc^*$$
$$\text{HWDATA} = dc\,;\,Data_1\,;\,dc^*$$
$$\text{HREADY} = H\,;H\,;\,dc^*$$

(where dc stands for "don't care" and H for "high"). These specifications, however, are too rigid to capture the instances of write transfers in the burst sequence (diagram 5-12). The

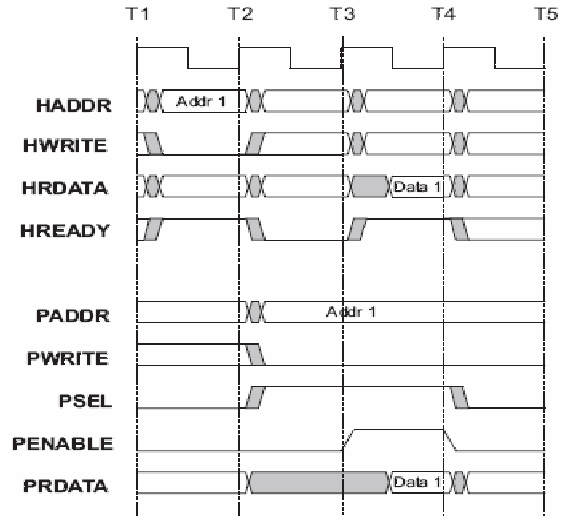

Figure 5-9 Read transfer to AHB

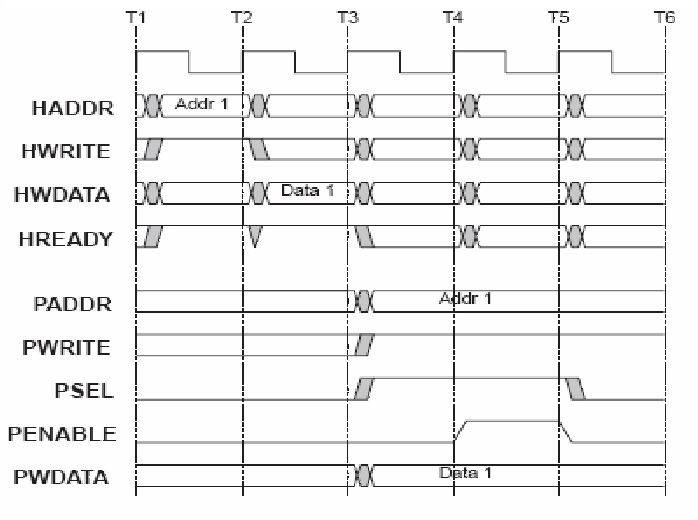

Figure 5-11 Write transfer from AHB

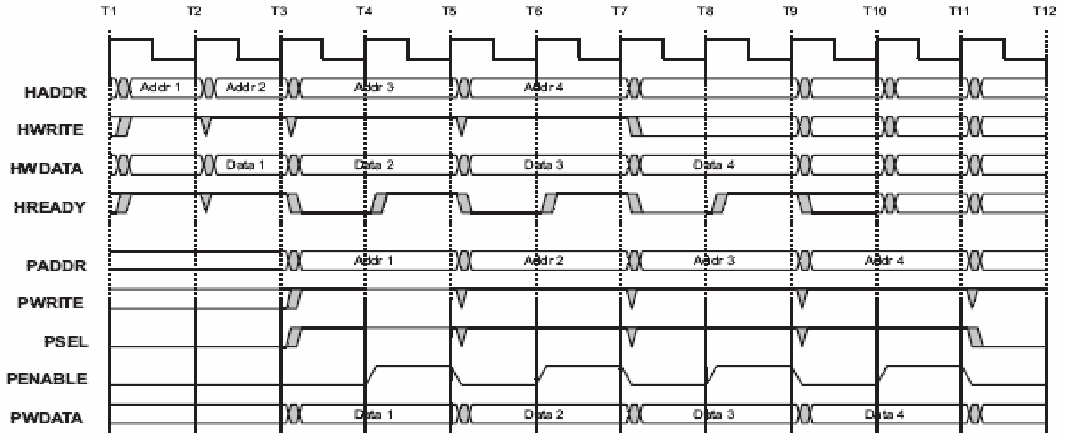

Figure 5-12 Burst of write transfers

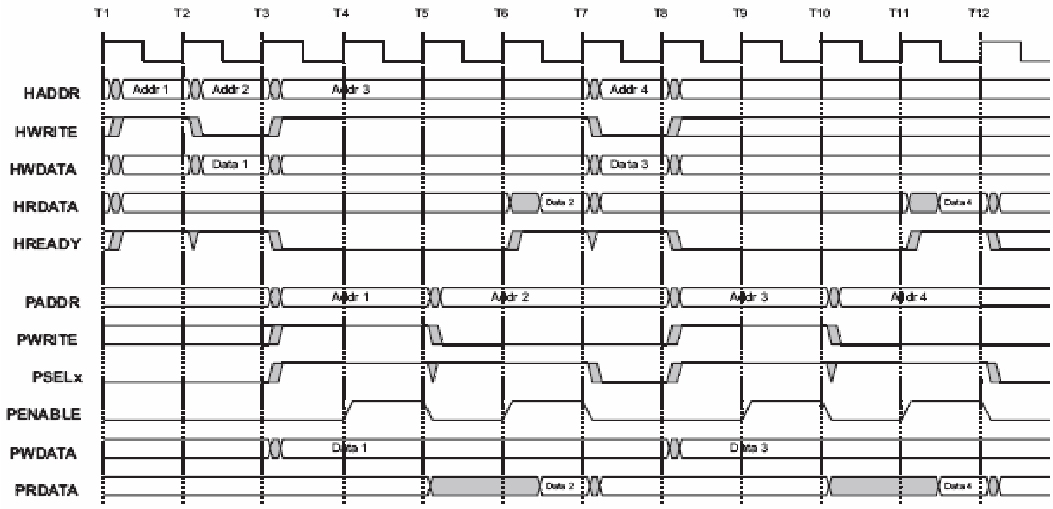

Figure 5-13 Back to back transfers

Fig. 1. Diagrams from the AMBA-2 specification [1] (pages 5-14 – 5-18) showing single read and write transfers (top), burst sequence of writes (middle), and sequence of read and write transfers (bottom). The lower two diagrams consist of pipelined instances (compositions) of the behaviors in the two diagrams in the top row. The figure numbers listed under the diagrams are from the original specification; we refer to the original numbers in the rest of the paper.

regular expressions require the *Data* value on HWDATA to occur in the next clock cycle after the *Addr* value on HADDR, but that does not happen for $Addr_3$/$Data_3$. Each of $Addr_3$ and $Data_3$ may also extend beyond the single clock cycle shown in the basic diagram. This suggests not equating clock cycles with steps in the regular expression. In similar vein, the regular expression for HREADY contains two cycles at high value, with the first synchronized with $Addr_1$. In the burst diagram, the HREADY high value for $Addr_2$ occurs simultaneously with $Addr_3$ on HADDR (due to pipelining), and only lasts a single clock cycle. Rewriting the regular expression for HREADY to expect only a single high value would address the latter problem, but not the synchronization problem.

As these examples illustrate, the synchronization implicit in regular expressions running concurrently is fundamentally limiting for protocols in which relative position of terms matters more than precise cycle counts. Regular expressions naturally capture individual signals, but their composition needs to be more nuanced. Modeling potential glitches exacerbates the limitations of regular expressions. A combination of pipelined regular expressions with the more flexible alignment of events natural in timing diagrams seems promising. Specifically, we decouple concatenation from clocking using synchronization and ordering constraints inspired by timing diagrams to relate atomic expressions across regular expressions. The resulting combination of inter-signal and cross-signal specification, exemplified by timing diagrams, inspires our term "two dimensional regular expressions".

III. 2D Regular Expressions

We develop our 2D regular expression from the ground up, working with the diagrams in figure 1. The basic read protocol (diagram 5-9) suggests the atomic terms needed to capture signals: high and low bit values (seen on HREADY), denoted H and L, respectively; named bus values (such as $Addr_1$ on HADDR), denoted by variable names; unnamed but stable values (on HADDR after T_2 and HREADY after T_4), denoted by *sd* (for "stable but don't care") and potentially unstable values (the shaded areas on all signals), denoted by *uv* (for "unstable value"). We use the standard operators of ; for concatenation, + for choice, and * for iteration, plus Oliveira and Hu's @ for pipelined concatenation. The diagrams we have studied so far rarely, if ever, nest uses of the * operator, so the expressions for signals tend to be relatively simple.

Using these atomic terms, the HADDR signal in the read diagram appears to correspond to

$$sd\,;\,uv\,;\,Addr_1\,@\,uv\,;\,sd\,;\,uv\,;\,sd\,;\,uv\,;\,sd.$$

This expression expects three (potentially distinct) stable bus values to occur on HADDR after $Addr_1$ appears. Furthermore, the corresponding $Data_1$ value on HRDATA is expected after two of those stable values have appeared. In the read/write transfer sequence (diagram 5-13), the second transfer (starting with $Addr_2$) is a read transfer and should be consistent with this expression. There, $Data_2$ occurs after only one *uv* value on HADDR. The proposed HADDR expression is therefore overly specific relative to how the transfer gets instantiated in sequence. The essence of the problem is that the basic read protocol must specify the *potential* for unstable values, whereas the semantics of regular expressions *requires* the unstable period to occur. To retain flexibility for composition, then, *the regular expressions for a signal should capture only required behavior*, leaving optional behavior to auxiliary annotations. We accordingly eliminate the *uv* values from the regular expression. For flexibility, we also exploit * to avoid fixing the number of stable regions that will follow $Addr_1$. The resulting specification of HADDR is $sd\,;\,Addr_1\,@\,sd^*$. Figure 2 shows the full expression for the basic read diagram, and is explained in the rest of this section.

While our treatment of HADDR suggests that *uv* values should never appear in the regular expressions for signals, PRDATA shows an example where *uv* is appropriate. On PRDATA, the long period of instability is a standard part of the protocol, and should be included in the regular expression. We capture PRDATA as $sd\,;\,uv\,;\,Data_1\,;\,sd$. The decision of whether or not to model a shaded region as *uv* depends on whether the depicted period of instability will necessarily occur.

The protocols require certain atomic terms to happen either concurrently or in particular orders. In the core read protocol (diagram 5-9), $Data_1$ appears on HRDATA after the rising transition on HREADY, and the fall of HREADY aligns with the end of when $Data_1$ is stable. The following two annotations capture this, where S_{Ti} denotes the i^{th} concatenation operator (; or @) in signal S (counting from 1):

Order(HREADY$_{T3}$, HRDATA$_{T1}$)
Event(HREADY$_{T4}$, HRDATA$_{T2}$)

These have the obvious semantics: Event requires its arguments to be concurrent and Order requires the second term to occur strictly after the first. Optional numeric arguments to Order give lower and upper bounds on the time between the events. Additional Event specifications align portions of HADDR, HWRITE, and HREADY as shown in figure 2.

This leaves handling the potentially unstable regions (glitches) in the signals. The unstable regions omitted from HADDR occur around transitions. The regular expressions defining the signals suggest that transitions consume no measurable time: expression $H;Addr_1$ has $Addr_1$ stable in the time unit immediately following H. To support glitches, we must relax this. Our transitions have distinct start and end times that may be separated by more than one unit of time. Often, the fact that transitions may take time is more important than the amount of time that they take. Annotation GlitchTrans(S_{Ti}) indicates that the i^{th} transition on signal S may take multiple units of time. Unlike the Event and Order annotations, GlitchTrans annotations may be ignored (desirable for analyses that abstract away timing): each GlitchTrans extends the set of traces that satisfy the expression, whereas the other annotations restrict the set of satisfying traces.

The read diagram contains two additional pieces of information about glitches: many glitch regions span the same period of time, and that period lies within the high clock period (the unnamed signal at the top of the diagram). Event annotations

HADDR $= sd; Addr_1 \,@\, sd^*$ [* allows multiple transfers]
HWRITE $= sd; L \,@\, sd^*$
HRDATA $= sd^*; Data_1 \,@\, sd$
HREADY $= sd; H; L; H \,@\, sd$

Event(HADDR$_{T1}$, HWRITE$_{T1}$, HREADY$_{T1}$)
Event(HADDR$_{T2}$, HWRITE$_{T2}$, HREADY$_{T2}$)
Event(HRDATA$_{T2}$, HREADY$_{T4}$)
Event(HRDATA$_{T1-first}$, HREADY$_{T3-first}$)
Order(HRDATA$_{T1}$, HREADY$_{T3-last}$)

ClockSignal(CLK)
GlitchTrans(HREADY$_{T1}$)
Constrain(HADDR$_{T1-first}$, HADDR$_{T1-last}$, CLK $= H$)
[similar constraints for remaining glitch regions]

Fig. 2. 2D regular expression for the basic read transfer protocol (AMBA diagram 5-9). The ClockSignal annotation indicates that the named signal oscillates between high and low values as a clock is expected to do.

align the endpoints of glitch regions ($S_{Ti-first}$ and $S_{Ti-last}$ refer to the start and end times of the i^{th} transition on signal S). Containing these regions within positive clock periods is harder with the annotations given so far. Order constraints require naming the clock transitions, which is difficult since the number of clock cycles used in a composed transaction may vary. Instead, we introduce an annotation Constrain(e_1, e_2, $s = v$) that requires signal s to hold value v between events e_1 and e_2. An example appears in figure 2. An Order annotation on the start and end times could bound glitch durations.

This discussion of glitches, events, and order constraints potentially masks the more substantive points about using regular expressions to effectively model the AMBA-2 protocols compositionally. The protocols as shown in the diagrams capture different information at different granularities of clocking: the glitch regions fall between the clock cycles governing the logical heart of the protocols. Multiple clocking granularities are also useful for capturing the essence of high-level signal behaviors: a value that spans only one clock cycle in the basic protocol diagram may span several in a composed sequence, while corresponding to a single logical concept in the protocol. Other AMBA protocols (such as multi-address burst commands from a bus master [AMBA-2 figure 3-6, not shown]) contain signals whose values change upon events in other signals. Decoupling clocking from the notion of time implicit in the regular expressions seems essential to handling these problems cleanly; some system of synchronization and ordering constraints is needed to make up the difference. Various notations suffice, from explicit Event and Order constraints on all related pairs to statements that assign signals to clocks and rely on the usual concurrent semantics of regular expressions. The differences are syntactic and irrelevant for this paper.

IV. Perspective and Discussion

This paper describes the early stages of a project to formalize interface specifications while capturing the relationships between their constituent protocols. AMBA-2 develops its protocols iteratively, refining basic behaviors through successive sections of the document. Using the framework outlined here, AMBA-2 figures 5-12 and 5-13 are instances of the protocols shown in figures 5-9 and 5-11; only the latter two (the basic protocols) must be specified explicitly. We are trying to identify composition constructs that enable this sort of reuse across the entire specification without sacrificing readability or ease of abstraction for different analysis tasks. Two principles have emerged from our work so far: basic protocol specifications must treat clocking and signal values sufficiently abstractly to support pipelined composition, and clocking overall must be decoupled from concatenation to enable specification at different levels of time abstraction.

Pipelined regular expressions make it feasible to use regular expression-like notation for some protocol compositions, but lack the fine-grained timing control needed to compositionally specify AMBA-2. Timing diagrams handle multiple time granularities well, but can be a bit too concrete in the face of arbitrary * operators and cross-signal constraints that depend on sequences of events. We could add a pipelining notation to timing diagrams to handle the examples in this paper, but that would be premature. Other AMBA-2 protocols that extend the core operators (such as those for error handling) demand different composition operators. We want to identify these operators before choosing a notation. Aspect-oriented programming [5] offers composition operators for fine-grained behavior extensions that are promising for the additional protocols. Balancing the rich information structure of the diagrams while retaining sufficient abstraction and readability is an important area for future work.

We are also applying our constructs to AXI, HyperTransport, and the OVL assertion language (also specified largely diagrammatically). The latter effort is promising, but the former two lack the explicit iterative development style of AMBA-2. We are presenting this work as a short paper in hopes of gaining early insight into what determines the high-level structure of an interface protocol specification, and what kinds of high-level structures lead to designer-friendly specifications.

Acknowledgements: Mike Gordon proposed studying the structure of timing diagrams in bus specifications via AMBA-2. NSF grants CCR-0132659 and CCR-0305834 funded this work. The reviewers provided several thought-provoking comments.

References

[1] ARM Limited, "AMBA specification (rev 2.0)," May 1999, http://www.arm.com/products/solutions/amba2overview.html.
[2] K. Fisler, "Timing diagrams: Formalization and algorithmic verification," *Journal of Logic, Language, and Information*, vol. 8, pp. 323–361, 1999.
[3] N. Amla, E. A. Emerson, and K. S. Namjoshi, "Efficient decompositional model checking for regular timing diagrams," in *IFIP Conference on Correct Hardware Design and Verification Methods*, 1999.
[4] M. T. Oliveira and A. J. Hu, "High-level specification and automatic generation of IP interface monitors," in *International Conference on Design Automation*. ACM Press, 2002, pp. 129–134.
[5] "Aspect oriented programming (article series)," *Communications of the ACM*, vol. 44, no. 10, Oct. 2001.