CSEE W4823x
Prof. Steven Nowick

# Project #2: The Minimalist CAD Tool and Asynchronous Controller Design

Handout #34d
December 2, 2016

This small homework is part of your final submission for the second course project. It is *worth 2%* of your final grade (the remainder of project #2, Handout #34, is worth 18% of your final grade).

This assignment is also due on **Monday, December 12 (last day of classes), at 4pm**. It should be submitted to the CA's with same directions as for the RTL assignment.

*Note:* A correct answer without adequate explanation or derivation will have points deducted. To get full credit, (a) write legibly/type, and (b) show all work (label relevant items, show deriviations, include explanations).
**Working in Groups.** You are allowed to do this homework in a group-of-two. If so, it should be the same group you are in for the rest of the project. You both get the same grade. However, solo homeworks are also allowed.

0. (0 points (don't hand in!)) **Asynchronous Design: Introduction to the MINIMALIST CAD Tool.**
   This problem will introduce you to the MINIMALIST CAD package, for the design of asynchronous controllers.

   *Getting Started: Basic Tutorial on the MINIMALIST CAD Tool.*
   First, review Handout #38 (pp. 1-36), on asynchronous burst-mode controllers.

   Also, see the top-level class web page, *Minimalist Tool Setup,* which tells how to access the tool either through the Embedded Systems Lab (locally or remotely) or via download (for Linux machines only).

   Then *do the tutorial* on Handout #39, pp. 1-34, which covers examples #1, #2A and #2B. Compare your results at each step to those listed in the tutorial slides.

   *Do not hand in your results! This tutorial is simply for you to learn the tool.*

1. (25 points) **Burst-Mode Asynchronous Controllers: Alternator Component (part 1).**
   You are to design a burst-mode specification for a *sequential alternator component,* and then synthesize it.

   The alternator component, at the block level, looks similar to the Tangram (i.e. Philips) "mixer" of Handout #39, examples #2A and #2B, but with reversed channel structure: it has one left channel (A) and two right channels (B, C). Each channel is implemented with a pair of wires: channel A consists of input $AR$ and output $AA$; channel B consists of output $BR$ and input $BA$; and channel C consists of output $CR$ and input $CA$. Each channel uses a 4-phase handshaking protocol, hence all wires are initially 0.

   The component receives its first request (AR+) on left channel A. It then completes a 4-phase handshake on right channel B (BR+, BA+, BR-, BA-), then completes the remainder of the 4-phase handshake on channel A. When the next request (AR+) arrives on left channel A, a similar protocol occurs on the C channel (followed by completion of the remainder of the handshake on channel A). This process repeats. Effectively, activations on left channel A alternate between producing channel communication on B vs. channel communication on C.

   Your unoptimized design (like the unoptimized Tangram mixer #2A) should have *only one input change for each input burst.*

   *Do the following:*

   (a) *Draw a top-level block diagram of the component,* showing the three channels and label all the wires.

   (b) *Draw a burst-mode specification for this component,* clearly labelling the states and arcs.

   (c) *Synthesize an unoptimized burst-mode controller with Minimalist.* In particular, follow the steps of Example #1 of the tutorial, but on your new specification: (i) enter the specification in a text .bms file; (ii) display it with 'bms2ps' command; (iii) synthesize it with *three scripts:* minimalist-basic, minimalist-area, and minimalist-speed, and generate the text output results; and (iv) display the results using “plot_nand”. *Hand in your .bms file of part (i) and synthesized text results summary of part (iii).*

*Note: Displaying PS figures.* As part of the above problem, you are asked to use “bms2ps” to display the burst-mode specification in part (c)(i) in graphical format. The command ’bms2ps’ only generates the corresponding Postscript (.ps) file. You can view the file by running a utility like Ghostview (gv) or equivalent Postscript previewer. Or, you can convert it to PDF using ’distill’. For example, if you you have entered your specification in file “foo.bms”, the command “bms2ps foo.bms” will generate the file foo.ps. Then use gv to display it directly or convert to PDF first.

2. (25 points) **Burst-Mode Asynchronous Controllers: Alternator Component (part 2).**
You are now to redo the above problem, but for a more parallel version: a *concurrent alternator component.* Your optimized design (like the concurrent Tangram mixer (#2B) in Handout #39) should aim for a minimal number of arcs in the burst-mode specification. The result is a design which allows concurrent operation on both interfaces, and hence has better system performance.

   *What to Do:* complete parts (a), (b) and (c) of the previous problem, but for this new optimized design.

   *Guidelines:* similar to the concurrent Tangram mixer, the goal in designing your burst-mode specification is to have concurrent activation of left and right interfaces when possible. Certain restrictions should be followed: (i) the right interface must have entirely completed its handshaking protocol before generating the final AA- event to the left interface; and, (ii) as usual, whenever a channel produces an output event (high or low), the specification must be able to receive an immediate subsequent input event: that is, a channel is assumed to support continuous activation, so your specification must be able to accommodate a subsequent input event immediately after it issues the output event.

3. (25 points) **Burst-Mode Asynchronous Controllers: Channel Merge Component (part 1).**
You are to design a burst-mode specification for a *channel merge component,* and then synthesize it.

   The component, at the block level, looks similar to the Philips Tangram “mixer” of Handout #39, examples #2A and #2B, but has different behavior. The component has two left channels (A, B) and one right channel (C). Each channel is implemented with a pair of wires: channel A consists of input $AR$ and output $AA$; channel B consists of input $BR$ and output $BA$; and channel C consists of output $CR$ and input $CA$. Each channel uses a 4-phase handshaking protocol.

   The component synchronizes on requests from both input channels (A, B), then activates output channel C, and the completes the handshaking on all interfaces. In more detail, the component first waits to receive requests on both of the left channels (AR+, BR+). These requests may arrive in any order or time. Once both are received, it completes an entire transaction (i.e. 4-phase handshake) on right channel C. Finally, once the C transaction is complete, it concurrently completes the handshaking communication on both of the input channels: issues simultaneous acknowledgment output transitions AA+, BA+; then waits for both A and B to lower their requests AR and BR; then finally simultaneously lowers the acknowledgment output transitions on AA and BA. Once completed, this entire process can repeat.

   *Do the following:*

   (a) *Draw a top-level block diagram of the component,* showing all the channels, and labelling all their wires.

   (b) *Draw a burst-mode specification for this component,* clearly labelling the states and arcs.

   (c) *Synthesize a burst-mode controller with Minimalist.* In particular, follow the steps of Example #1 of the tutorial, but on your new specification: (i) enter the specification in a text .bms file; (ii) display it with ’bms2ps’ command (see “Note” above); (iii) synthesize it with *three scripts:* minimalist-basic, minimalist-area, and minimalist-speed, and generate the text output results; and (iv) display the results using “plot_nand”. *Hand in your .bms file of part (i), a printout of your specification from part (ii), and synthesized text results summary of part (iii).*

4. (25 points) **Burst-Mode Asynchronous Controllers: Channel Merge Component (part 2).**
Redo the previous problem on the channel merge component, but now specifying and synthesizing a more concurrent (i.e. more parallel) version.

   *Hint:* to understand how a specification can be optimized to support greater parallelism, review the un-optimized version of the Tangram channel "mixer" component in Handout #39, example #2a, and then compare it against the more concurrent optimized version presented in example #2b.

   *Guidelines:* Several guidelines must be met to ensure a correct design during your optimization. Of course, output channel C cannot be activated until both A and B requests (high) have arrived. Also, four-phase handshaking must continue to be followed on each interface. In addition, the final lowering of the left acknowledgments (AA-, BA-) can only occur after the entire four-phase handshaking protocol is complete on the output channnel C. Finally, as usual, whenever a channel issues an output event (high or low), the specification must be able to receive an immediate subsequent input event: that is, a channel is assumed to to support continuous activation, so your specification must be able to accommodate a subsequent input event immediately after it issues an output event.

   *What to do:* Repeat parts (a), (b) and (c) above, but for your more optimized new design.