μSystems Research Group

School of Electrical and Electronic Engineering

Modelling Digital Systems using Behavioural Fragments

J. Beaumont

Technical Report Series

NCL-EEE-MICRO-TR-2015-194

January 2015

Contact: j.r.beaumont@ncl.ac.uk

Supported by EPSRC grant EP/L025507/1

NCL-EEE-MICRO-TR-2015-194

μSystems Research Group
School of Electrical and Electronic Engineering
Merz Court
Newcastle University
Newcastle upon Tyne, NE1 7RU, UK

http://async.org.uk/

Modelling Digital Systems using Behavioural Fragments

Jonathan Beaumont

I. Introduction

This technical report serves as my research proposal, and is a discussion of the work I have been doing since I started my post graduate research in September. The following information is the start of what I plan to continue advancing upon during the remainder of my post graduate degree.

A. Motivation

When designing a digital system, models are used to design and test how the system will operate, and ensure that signals transition in the correct orders and that the function the system is designed for will be carried out correctly. Designing a full system can be a complex and lengthy process, requiring several stages of tests and re-designs to add further functionality and for finding and correcting errors. It is possible to separate a full system into several scenarios which can be designed separately and combined but depending on the complexity of each scenario this may not change the length of the process. These models can then be synthesized to a set of logic gates which will implement the model designed for the system.

Using software, it is possible to automate the process of creating models. Automation of the design process can reduce 1. the length of time to create a large model and 2. the chance of errors in a design which, in turn, reduces the length of time needed to find and correct errors. Overall this can reduce the design time of a system.

B. Importance

In industry, the time from conception of a device to production and being put on sale, known as time to market, can be very lengthy, from a period of several months to a period of years. Designing a device can be one of the causes of an increase on the time to market, and in the case of a digital system, the more complex a system, the more difficult it is to design and test. If the process can be completed quicker it means the time to market can be reduced. The time saved can be used to better the product, for example by adding features, or making the design smaller.

C. Contribution

The automation process I will discuss doesn't focus on creating the full model all at once, but instead focuses on functionality requirements, safety constraints, communication protocols, etc. of the system. Requirements can be described as smaller models themselves, which I call *fragments*, and these can then be combined to compose a full model. If there are multiple scenarios of a system these can each be composed of fragments, which may be different depending on the requirements of each scenario. All scenarios can then be combined to create a full system model. This process I will discuss in this paper.

II. Design Flow

Along with the automation for this method of design comes some changes which are necessary in order for software to work correctly and return a correct model. I will explain the design flow for this method, and use it throughout the paper as an example. Each part is important for its own reasons and the following explanations will explain why each part is necessary for the design of digital systems using fragments.

- Decide on Scenarios - If the system is large enough, splitting it into scenarios can allow each scenario to be composed of fragments separately. In some cases, scenarios can be made for operating modes in systems. Systems which could take advantage of this include power-elastic systems [11] or microprocessor systems which adapt to operating conditions [7]. Each operating condition, for example power saving mode or high performance mode, will have different functionality and as such, their models will be different. Each of these modes can be modeled separately. In many cases, a model will have similarities between scenarios, but differences which mean they cannot all be composed of the same fragments. Finding these differences is useful for finding fragments.
- Find Fragments - When scenarios have been found, finding fragments for the scenarios can be done. Each fragment should represent a different requirement of the system, and will be understandable when separated from the rest of the system. Because of this, it can be reused across multiple scenarios, similar to a sub-routine when writing software - it may be useful in several different areas, and thus, separating this avoids repetition. Fragments and scenarios could be confused, but for this example, fragments are used to describe the very lowest form of requirements for the systems, where as scenarios describe one of the major operations of a system which can be separated from other operations for *composition*.
- Compose each scenario - When fragments have been found and created, these can be composed to create complete models for each scenario.
- Combine composed scenarios - With a model for each scenario, these can then be combined, again using software.
- Full Model - The combined scenarios will provide a model for the full system.

- Synthesis - The full system model can be *synthesized*, producing a list of logic gates used to create a circuit.
- Circuit - This can be created from the list of gates from the synthesis of the full model. This circuit will perform the functions the model has been designed to control.

Throughout the design example, I will be producing fragments and models as both Finite State Machines (FSM) [4] and Signal Transition Graphs (STG) [12], as well as explaining how the fragments are composed. Both of these types of models have the capability of being composed of fragments, but the models produced differ somewhat, and I will discuss these differences.

III. Decide on Scenarios

Specifications of digital microcontrollers will usually describe several possible operations which will happen separately from one another, and as such, deciding scenarios for this design flow can be done by splitting each of these operations into multiple separate descriptions, each one a scenario in itself.

As an example for this paper, I will use a simple buck control circuit [10] which is designed to control the voltage and current in a circuit, by switching on or off a p-type transistor which increases the voltage, or an n-type transistor which drains current from the circuit. These transistors cannot be switched on at the same time, but which one is on at which time is determined by several signals from the circuit. The buck controller itself has two output signals, each switches one of the transistors (see Figure 1). The signals and their descriptions can be found in Table 1.

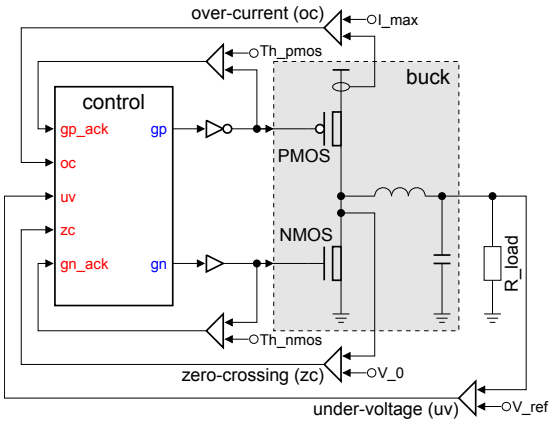

Figure 1. Basic Buck controller (illustration from [10])

From these signals, and discussing the operation of the circuit, we can deduce that there are three scenarios:

- UV without ZC - In this scenario, there is no zero crossing involvement. The initial state is after over current has been dealt with and before under voltage has signaled. When under voltage signals, the n transistor is switched off by setting gn low, which is followed by gn_ack going low. gp is then set high to switch on the p-type transistor, which is acknowledged by gp_ack. After this under voltage will go low when the circuit voltage has returned to normal. Over current can then signal, which is responded to by switching off the p transistor by lowering gp, which will cause gp_ack to go low. gn will then be set high and when the n transistor is switched on, gn_ack will go low. At this point, the current will drain, and oc will go low when the current is low enough.
- UV before ZC - Here, zc is involved, but only after uv has triggered. In this case the initial state is the same as the previous scenario and when under voltage is high, the same signal changes occur as with the previous scenario, turning the n-type transistor off and the p-type transistor on. Once uv has been signaled, zc can then concurrently transition from low to high and then low again, before under voltage can transition low again. At this point oc can signal and is dealt with in the same way as the previous scenario.
- UV after ZC - Zero crossing has large involvement in this scenario. The initial state here is again just after over current has been dealt with, but instead of uv, zc signals first. When zero crossing signals, the n transistor is turned off, but the p transistor is not turned on until under voltage signals. When uv goes high, the p transistor is turned on, to increase the voltage in the circuit. Before under voltage can go low, zc must go low. After this, oc can signal and be dealt with the same way as the other two scenarios.

IV. Find Fragments

Finding fragments is a process of knowing the systems signals, and understanding what each one represents within the system. When this is understood, it is a case of understanding how each of the signals can affect others and combining them in small models.

With several scenarios prepared, we can now start working out requirements for each one which can then be modeled as fragments. In this example there will be similarities between the three scenarios, but important differences.

A. UV without ZC

There are several requirements based on how the circuit works for this scenario. For example, when gp transitions, gp_ack transitions as well to acknowledge that the p-type transistor has been switched on. This is a handshake protocol. The same applies to gn and gn_ack. The fragments for these protocols as FSM and STG are as follows:

Signal Label	Signal Name	Notes
uv	Under Voltage	Input signal, shows when the circuit voltage is low
oc	Over Current	Input signal, shows when circuit current is high
zc	Zero Crossing	Input signal, shows when the voltage and current are within normal range
gp_ack	GP Acknowledge	Input signal, acknowledges when the p-type transistor is turned on
gn_ack	GN Acknowledge	Input signal, acknowledges when the n-type transistor is turned on
gp	GP	Output signal, used to switch the p-type transistor on
gn	GN	Output signal, used to switch the n-type transistor on

Table I
SIGNALS USED IN THE BUCK CONTROL EXAMPLE

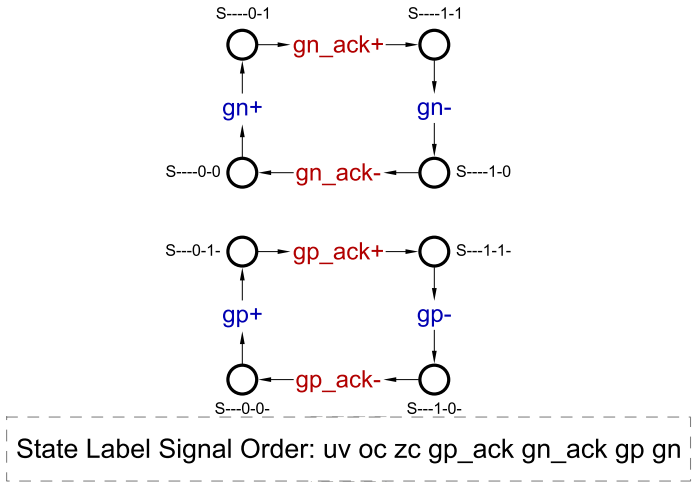

Figure 2. Handshake protocols for gn/gn_ack and gp/gp_ack as FSM

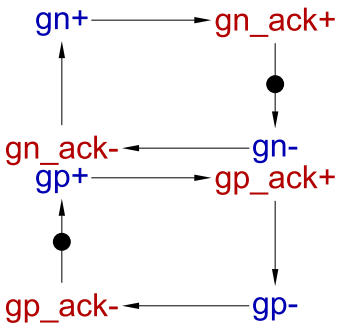

Figure 3. Handshake protocols for gn/gn_ack and gp/gp_ack as STG

Figures 2 and 3 show the handshake protocols, simply showing that, for example, when gn+ happens, gn_ack+ can happen, and gn- cannot happen before gn_ack+. The FSM and STG fragments are very similar in this case. FSMs are used to show what state a system is in - in this case what positions the signals are in - and what transitions can occur from this state. STGs show this in a different way, by abstracting away the state the system is in, but showing which signal transitions can occur and which transitions can occur as a cause of this transition. Thus, because these signals, either gn/gn_ack or gp/gp_ack, must transition in these orders, the FSM and STG models look similar.

The next fragments we can devise are based on under voltage and over current. These two signals cannot both be high at the same time, as only one of these circuit states can be dealt with at once. Here we can create mutexes to deal with this:

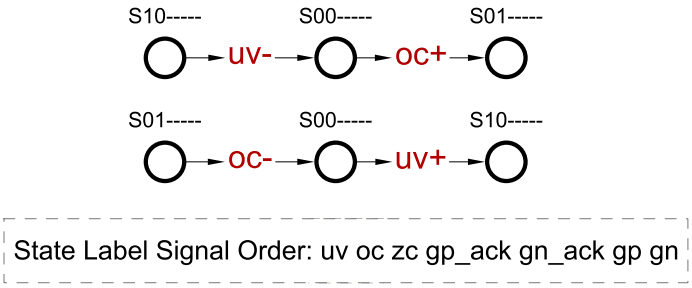

Figure 4. FSM mutex which aims to stop uv and oc signaling at the same time

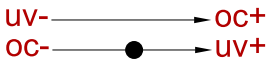

Figure 5. STG mutex which aims to stop uv and oc signaling at the same time

In Figures 4 and 5, it is shown in the two separate fragments allow oc to go high only when uv has been set low, and likewise, that uv can only signal when oc- has occurred. This will stop the model from having a case where both uv and oc are high at the same time, but it is possible for one to be high, or both to be low. The FSM in Figure 4 uses two arcs to show each mutex, where as the STG mutexes only have one. Again, because of the way STGs abstract states, it only uses an arc to show that uv- causes oc+, for example, but the FSM shows what the signal positions are with the state labels.

Some more mutexes can be discovered using the knowledge that, to avoid short circuits, both transistors cannot be switched on at the same time. We know that gn and gp control the n- and p-type transistors respectively, and we know that gn_ack and gp_ack are input signals to acknowledge that the n- and p-type transistors have been switched on or off, thus it can be

deduced that before gp can be set high, we need to know that the n-type transistor is off using gn_ack etc.

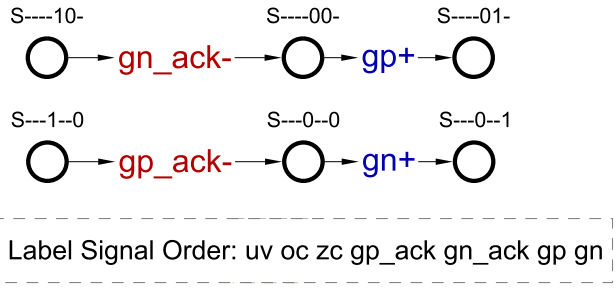

Figure 6. FSM mutex which stops both transistors turning on at once

Figure 7. STG mutex which stops both transistors turning on at once

The mutexes shown in Figures 6 and 7 will mean that only when the acknowledges for the one transistor is low, then can the other transistor be switched on. The FSM and STG models again have differing number of arcs, for the same reasons as with Figure 4 and 5. This is the same with all mutexes.

All the fragments so far have been necessary to avoid issues which could break the circuit. Now we need to add fragments which actually cause the correct change in the circuit to deal with either under voltage or over current. The basic operation for these is:

- Under Voltage - Switch off the n-type transistor and switch on the p-type transistor.
- Over Current - Switch off the p-type transistor and switch on the n-type transistor.

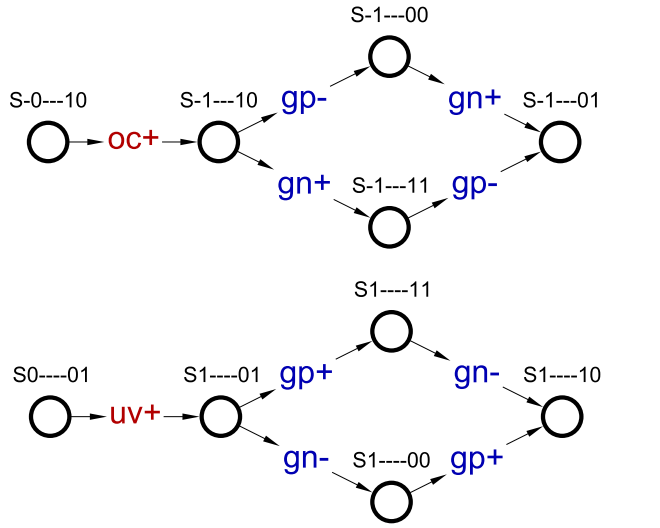

Figure 8. FSM function fragments for uv+ and oc+

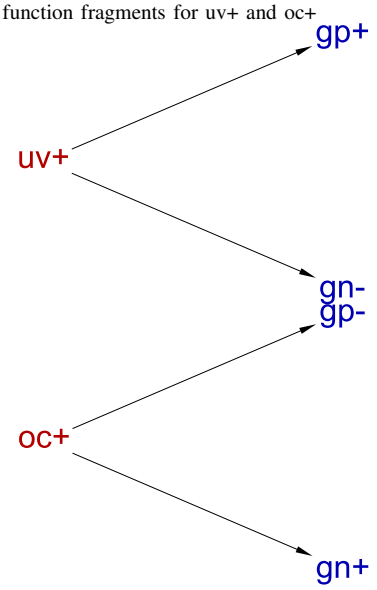

Figure 9. STG function fragments for uv+ and oc+

The FSMs in Figure 8 show multiple paths and repeated signal transitions. For example the top FSM shows that when uv signals by going high, then gp can go high and gn can go low, or gn can go low followed by gp going high. Because only one transition can occur at a time, it is necessary to account for these signals transitioning in either order, but as long as both occur it doesn't matter the order. However, because STGs show are more compact, Figure 9 represents the same as the FSMs in Figure 8, but rather than show the possible orders of signal transitions, it shows that the input signals, either uv+ or oc+, cause the respective transitions in gp and gn, and this shows that either order of gp and gn transitioning can occur.

The last fragments we need to find for uv without zc are those that react to switching of transistors. As discussed before, using the acknowledge signals we can tell which transistors are on or off, and using these we can decide which

states the acknowledges need to be in to cause uv or oc to go low. The signals need to be as follows:

- Under Voltage - n transistor will be off, so gn_ack will be low. p transistor will be on, so gp_ack will be high.
- Over Current - n transistor will be on, so gn_ack will be high. p transistor will be off, so gp_ack will be low.

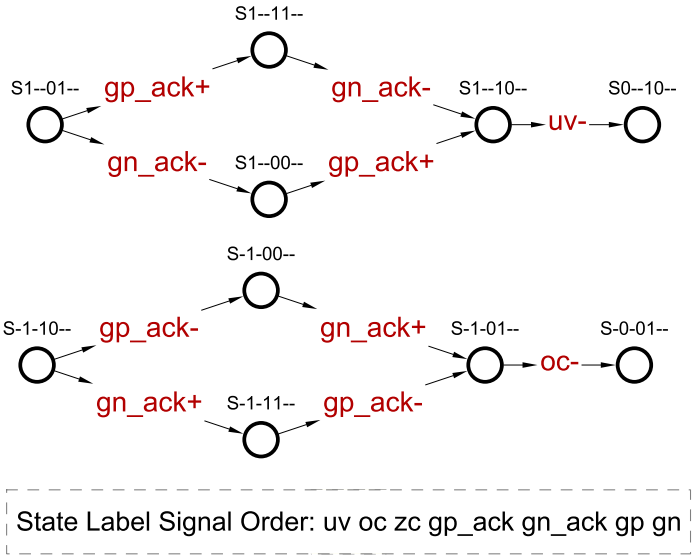

Figure 10. FSM reaction fragments for uv- and oc-

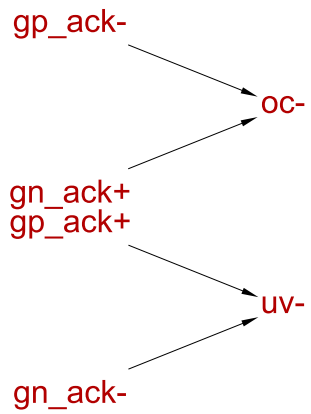

Figure 11. STG reaction fragments for uv- and oc-

Figure 10 shows the FSM fragments. As with Figure 8, there is two paths in each fragment to show every possible order of the signal transitions, which is not featured in the STG fragments in Figure 11. These fragments show that for uv or oc to go low, gp_ack and gn_ack must show the correct states of the transistors, n-type off - p-type on for uv- and p-type off - n-type on for oc-.

The fragments from Figure 2 to Figure 11 cover the specification for the buck controller circuit, for the scenario uv without zc. The other two scenarios may feature some of these, and may add some new fragments for their functionality.

B. UV before ZC

With this scenario, the zero crossing signal is included, so we need to include it in our analysis of requirements for the scenario fragments. If we use the same order as with the previous scenario, it is easier to find the similarities and differences between the scenarios.

Starting with gp/gp_ack and gn/gn_ack, these signals are used for the same purpose in this scenario as in the previous scenario, gp and gn used to switch on and off the p and n transistors, and gp_ack and gn_ack are used to signal the state of the transistor and acknowledge gp and gn. Therefore, the handshake protocols in Figures 2 and 3 will also be used for this scenario.

Under voltage and over current signals, as with the previous scenario, cannot be high at the same time, and therefore the mutex fragments in Figures 4 and 5 can be used for uv before zc. Similarly, to stop both transistors being switched on at the same time, we can use the mutex fragments from Figures 6 and 7 also.

The general function of the scenario hasn't changed either, because uv should signal before zc, uv will be dealt with in the same way, by switching off the n-type transistor, and switching on the p-type transistor, and the zc signal doesn't affect the operation of over current, so the functions in Figures 8 and 9 can still be used for this scenario.

Similarly, the reaction fragments in Figures 10 and 11 are not affected by zc so these can be included in the fragments for the uv before zc scenario.

Knowing that we can use the same fragments as the previous scenario, we have yet to include the zero crossing signal. From the specification of the scenario, we know that it needs to transition from high to low to high after uv is signaled. And before uv can transition low again, zc must have performed this transition.

Figure 13. STG fragment for zc interaction with uv

Figure 12 is the FSM for the functionality of zc within this scenario. Some of the state labels end with a letter, this is due to a Complete State Coding (CSC) conflict, where the state labels are the same for multiple states but each of these similarly named states has a different behaviour.[5]For example if this system is in state 1-0—-, there is no way of knowing whether zc has transitioned high-low or not. For this fragment, it is useful to keep the similar states labeled in this way, as the process of *composition* using this fragment will either remove the CSC conflict, or use it for the full composition of the FSM for the scenario.

The STG in Figure 13 doesn't show any CSC conflicts, it simply shows the order of signal transitions for this interaction between uv and zc, however it still contains a CSC conflict. This is caused in both the FSM and the STG by zc being set high, and then immediately set low (zc+ -> zc-), meaning that before and after this, the states will have the same encoding.

We now have the necessary fragments for the scenario uv before zc, which includes all the fragments in uv without zc (Figures 2 to 11) and one extra fragment, per modeling type, from Figures 12 and 13.

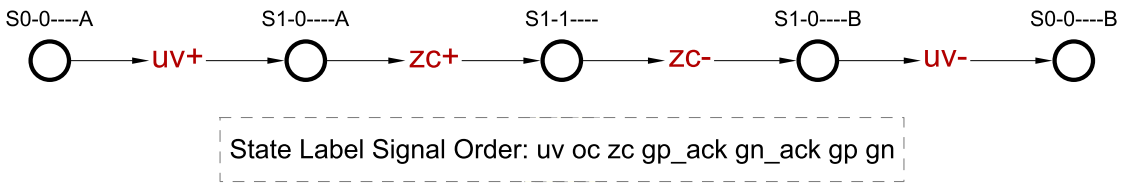

Figure 12. FSM fragment for zc interaction with uv

C. UV after ZC

The final scenario has some significant differences, particularly in the order of zc and uv. When analysing the requirements for this scenario, we need to ensure that zc is included in the correct manner, before uv is signaled.

Once again starting with the handshake protocols of gn/gn_ack and gp/gp_ack, the change of order of zc and uv doesn't affect these signals, which are still used to switch the transistors and acknowledge this. Therefore we can continue to use the protocol fragments in Figures 2 and 3 for this scenario.

The change of uv and zc in this case causes the mutexes from Figures 4 and 5 to be changed for this scenario. Zero crossing doesn't change the way that under voltage is dealt with, by turning on the p-type transistor, and when this has corrected the voltage in the circuit, uv will become low. This can then cause over current to signal, therefor the first mutex is still useable. However, because in the initial state, after over current has been corrected, zc will signal before uv in this scenario, so we need to change the second mutex.

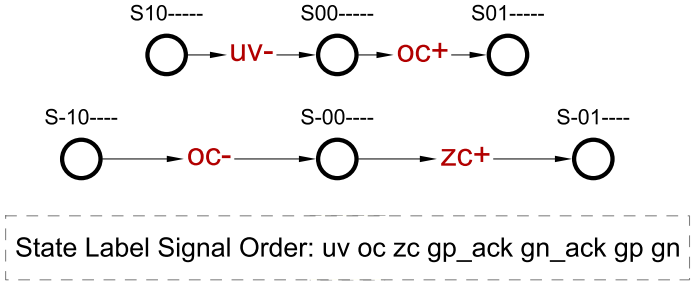

Figure 14. FSM uv, oc and zc interaction mutexes

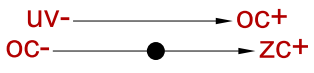

Figure 15. STG uv, oc and zc interactions

In Figure 14 and 15, these new mutexes stop oc and uv going high at the same time, as with the previous scenarios, but because zc signals before uv, we stop this zc signaling before oc is low to keep this interaction the same.

The mutexes used to stop both transistors being active at the same time, and thus avoiding short circuits, are not changed by the functionality of this scenario, there for fragments in Figure 6 and 7 will apply to this scenario.

The general function of correcting oc is unaffected by zc, so the function fragment in Figures 8 and 9 for oc will remain unchanged. However, as uv is not the first signal in this scenario, we need to correct this fragment.

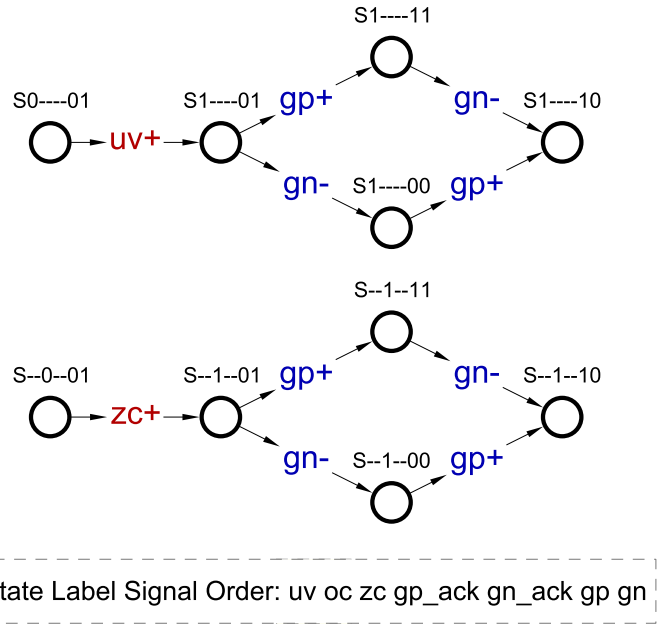

Figure 16. FSM fragment functions for oc and zc

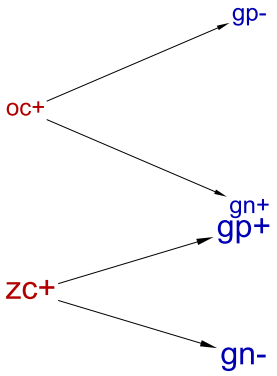

Figure 17. STG fragment functions for oc and zc

By replacing uv+ in the function by zc+, as seen in Figures 16 and 17, we account for uv signaling after zc.

Reactions of this scenario will remain unchanged, as they are still based on uv and oc and therefore Figures 10 and 11 can be used foe this scenario. However the zero crossing signal is required to be low before uv can transition low, and we need to account for this. We also need to include the interaction of zc and uv, where uv can go high after zc. As well as this, we need to ensure that gp does not transition high until uv has transitioned high, and once gp is high, zc can then go low. The following fragments are used to account for these.

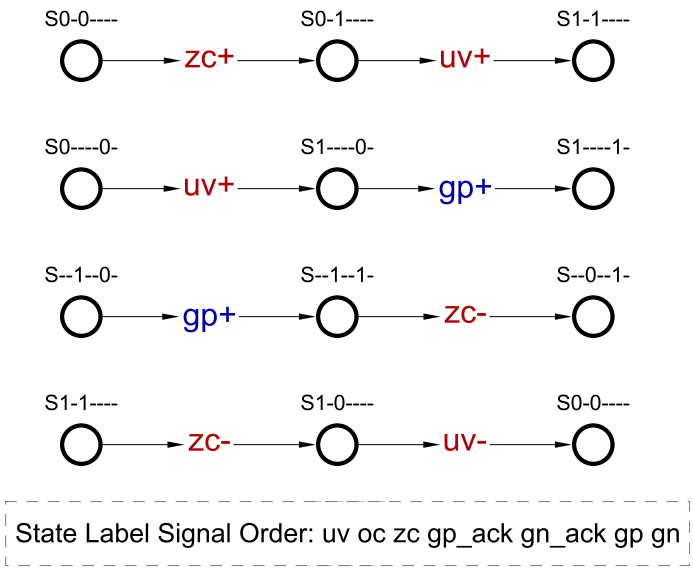

Figure 18. FSM fragments for uv after zc

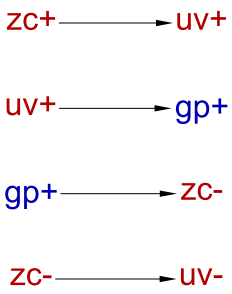

Figure 19. STG fragments for uv after zc

The fragments in Figures 18 and 19 will cause gp to transition high only when uv signals, after zc signals. When gp is high, gp_ack must follow it, according to the protocol, and zc must go low. When these have occurred, only then can uv go low.

The fragments to compose in this scenario are found in Figures 2, 3, 6, 7, 10, 11, 14, 15, 16, 17, 18, 19.

V. Compose Each Scenario

With fragments prepared for each scenario it is now possible to compose them into models. The FSM and STG methods of composition can both be automated by software, but have different methods of doing so, and I will begin by explaining how each composition method works.

A. FSM Composition

As of yet, there is no software to compose Finite State Machines from fragments, but there is methodology to do so, and this is what I will discuss in this section.

The fragments so far have featured labels with both numbers, 0 and 1 to represent the signal position, and hyphen characters '-'. The labels have been used to show which signals are changing at which points, ignoring the ones which do not change due to the fragments in question. For example, the gn/gn_ack handshake in Figure 2, the 0s and 1s within the states show the states of gn and gn_ack as they change. The hyphens represent the other signals which could be at any positions (don't care signals).

This idea is used for the composition method. If we know the initial state we can search the state labels in the fragments to see which signals are in similar positions. If multiple states have the same arc transition, we can combine these state labels to view more signals which should be in this position for transitions, and if the state doesn't agree with these states, then the transitions cannot occur. From here, we can add the arcs to the initial state to find new possible states. We then repeat this process with all these new possible states, and continue. If we know the final states as well, once all the possible states have been found we can find all paths from the start state to the final states, and remove all others, which will be loops, or dead ends.

Using uv without zc as an example, I will show the first few steps of composition using this method. As discussed before, the initial state is when over current has just been corrected, waiting for under voltage to signal. The n-type transistor is switched on, and the p-type is off. The first state is 0000101.

Original State	Possible Transition	Possible Next State
0000101	uv+	1000101
1000101	gn-	1000100
1000100	gn_ack-	1000000
1000000	gp+	1000010
1000010	gp_ack+	1001010
1001010	uv-	0001010
00001010	oc+	0101010

State Label Signal Order: uv oc zc gp_ack gn_ack gp gn

Table II
FSM Composition Table, showing states, and their possible next states after signal transitions

When looking at possible arcs, there are usually several states to combine and find that the original state from this Table doesn't comply, and therefore the arc will not be included from this state. Repeating the process, eventually, every state and its arcs will be followed and final states will be reached. If there are any other paths which do not reach a final state, these can be removed. Continuing this will return the full model for uv without zc.

B. STG composition

Unlike the composition of FSM, there is a software tool which performs efficient parallel composition of Signal Transition Graphs, known as PCOMP. This tool is used to efficiently combine models of subsystems to create a whole system model[2]. For example, using uv without zc, it would take one model, such as the first function in Figure 9,and using other fragments such as the protocols and mutexes which apply to these transitions and would create the following based on this:

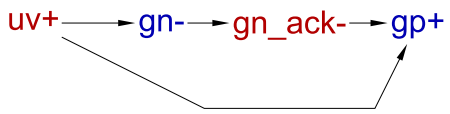

Figure 20. Example of STG Parallel Composition as performed by PCOMP

Figure 20 shows what the uv+ function will become when composed with PCOMP. This doesn't include all arcs, but for the example, it shows that it accounts for several of the mutexes and protocols, and includes the transitive arc between uv+ and gp+. I will show the rest of the composition in the following section.

C. UV without ZC composition

Figures 21 and 22 are the models for uv without zc, after being composed by the methods for their model types. Figure 21 shows the Finite State Machine, which after composition shows the cyclical nature of this model. The STG in Figure 22 also shows its cyclical nature. This STG has the correct functionality, the same as the FSM in Figure 21, but has many transitive arcs which do not affect the function of the STG, these are remaining from the parallel composition process, and make the model look quite convoluted. These however can be removed by a process called *resynthesis* which works by selectively composing STGs of components to obtain a smaller model [1], [3].

Resynthesis can be performed using software tools such as Petrify [5], and when passing this STG to petrify, the following STG is returned:

Figure 23 shows a much less convoluted STG, with the same functionality, but with the transitive arcs removed. Comparing the FSM in Figure 21 and the STG in Figure 23, we can see that the models look very similar for this scenario. We can see that under voltage is corrected by turning off the n-type transistor, and when this is acknowledge, the p-type transistor is switched on, which when is acknowledged can allow the uv signal to go low when the voltage has returned to a normal level. Over current is corrected by, quite oppositely, switching the p-type transistor off and the n-type transistor on when the acknowledge signal shows the p-type transistor is off. When the circuit returns to normal, oc can drop low which returns the model to it's initial state.

D. UV before ZC composition

With involvement from the zero crossing signal, we can now see how this changes the models, as it involves some concurrency which adds to the complexity of the models. For this and the final scenario, I will use STG models post resynthesis.

UV before ZC models, shown in Figure 24 and 25, show a large difference between the models and how they work. Both feature the same functionality; when uv signals, the n-type transistor is switched off, and following this the p-type transistor is switched on. Concurrently, zc goes high and then low, which must occur before uv can go low again. The over current signal is corrected in the same way as the previous scenario.

The concurrency is modeled entirely differently in both of these models, without changing the functionality. The FSM shows zc's transitions as possible during the gn, gn_ack, gp and gp_ack transitions to correct under voltage, which can happen at any point during this set of transitions, and therefore every state has the possibilities of zc transitions, or continuing to correct under voltage. Note that there are several states labeled with 'A' or 'B', which shows the CSC conflicts within this model. While this is an issue for a perfect working model, CSC is corrected during the synthesis step, which will be covered in a later section.

Signal Transition Graphs model concurrency rather than considering possibilities of transition orders, but by showing transitions in parallel, and using arcs to show which transitions must have happened before the next can, for example, the arc from gp_ack+ to uv- and the arc from zc- to uv- shows that both gp_ack+ and zc- must have occur before uv- can. Overall, STGs make for a much easier to understand model than that of an equivalent FSM when concurrency is involved.

E. UV after ZC composition

As with the previous scenario, the concurrent zero crossing involvement will cause differences between the FSM and STG models, but in this case there are two concurrent sets which involve a join between them.

Figures 26 and 27 are the models for UV after ZC. Again, both these models have the same functionality, but obviously represent it in different ways. Note that zc is the first signal which signals from the initial state, as required for this scenario. When this does signal, it switches the n-type transistor off, by setting gn low and waiting for the acknowledge of this, then concurrently, uv can signal, which must occur before gp can be set high and switch on the p transistor, which then must be acknowledged, as well as zc going low before under voltage can go low, signaling that under voltage has been corrected. Over current is corrected in the same way as the previous scenarios.

While much less convoluted, the FSM still has to account for several possibilities of signal transition orders, so when zc signals, it must be accounted for that, while the n-type transistor is being switched off, at some point uv will signal. gp will be set high only when the n-type transistor is signaled as off by gn_ack, and uv is high. The second set of concurrency shows that zc must go low and gp_ack must signal that the p-type transistor is on before uv can go low, but again these signals can transition in any order, which needs to be accounted for.

Because of the way STGs represent concurrency, Figure 27 doesn't have to account for signal order, it simply shows that, once zero crossing signals, that the n-type transistor will be switched off, and uv can also signal in parallel with this. gn_ack must be low and uv must be high before gp can be set high, and then in either order, zc must go low and gp_ack must be high before uv will be set low.

These two smaller concurrent sets have a join in the middle, which again shows an important difference between Finite State Machines and Signal Transition Graphs, which is that

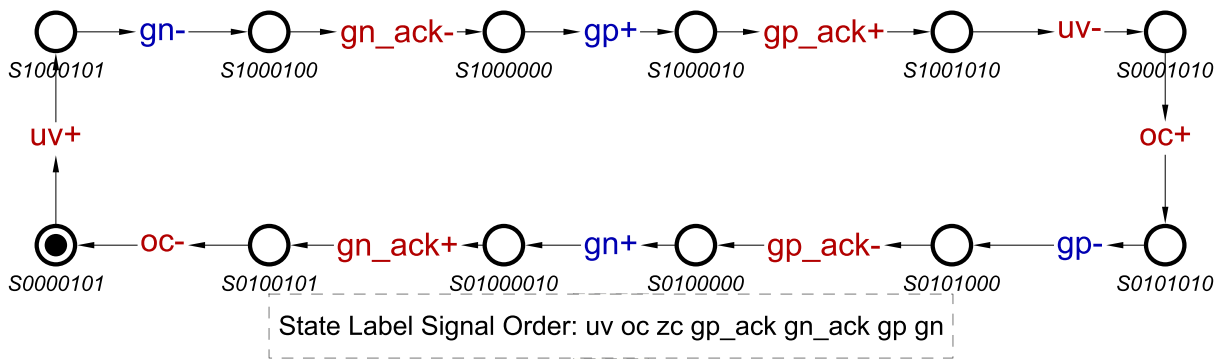

Figure 21. Full FSM model for UV without ZC

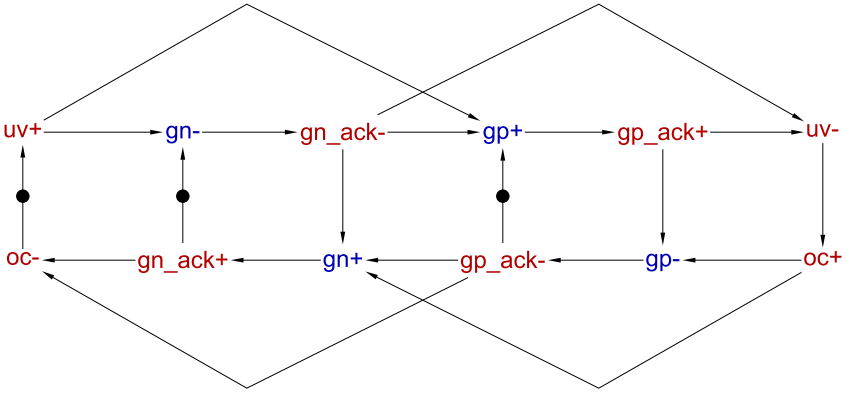

Figure 22. Full STG model for UV without ZC

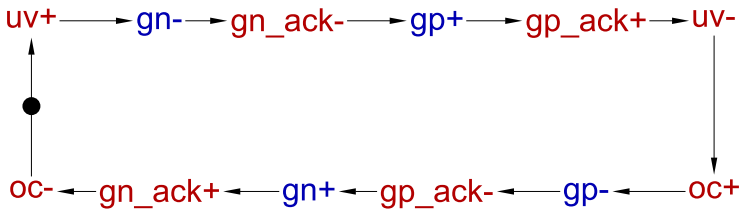

Figure 23. Full STG model for UV without ZC after resynthesis

the way they represent transitions and concurrency causes two rather different looking models, without affecting the actual functionality of these functions.

VI. Combine Scenarios & Full Model

With the scenarios all fully composed and correctly representing the functions we require from these scenarios, we can now move onto the next step of design, which is to combine these scenario models to create a full system model.

For combining these models, we can again automate the process using software tools. As with composition there is no tool to combine FSMs in this manner as of yet, however it is possible because the start states are the same, and therefore we can look at the following arcs and states and use their behaviors to determine how they will be connected in the full system model. Obviously in this example there are issues with CSC conflicts, and this will only increase in the full system model, but by judging the state behaviors, we can tell which scenario they are part of and keep them separated from each other, rather than have one state with all possible arcs to and from it, which would cause the scenarios to be connected, and functions to mix and not perform the tasks we require.

The full system FSM model, as seen in Figure 28, contains the under voltage elements from all three scenarios separated, but only one for the over current branch. This is because over current is corrected in the exact same ways in all three scenarios and therefore these can be combined. Viewing the state labels for the under voltage branches shows that there are several CSC conflicts, including some stats with labels 'D'. This is due to there being many states with the same labels, but different behaviors. Clearly there are huge issues with this model and CSC conflicts, but certain tools, such as Petrify can cope with CSC conflicts in order to synthesize the model correctly.

The STG model can be combined using Petrify, which will combine the scenarios together and look to remove any redundancies in the system, while keeping its requirements, and allowing only one scenario to be active at once.

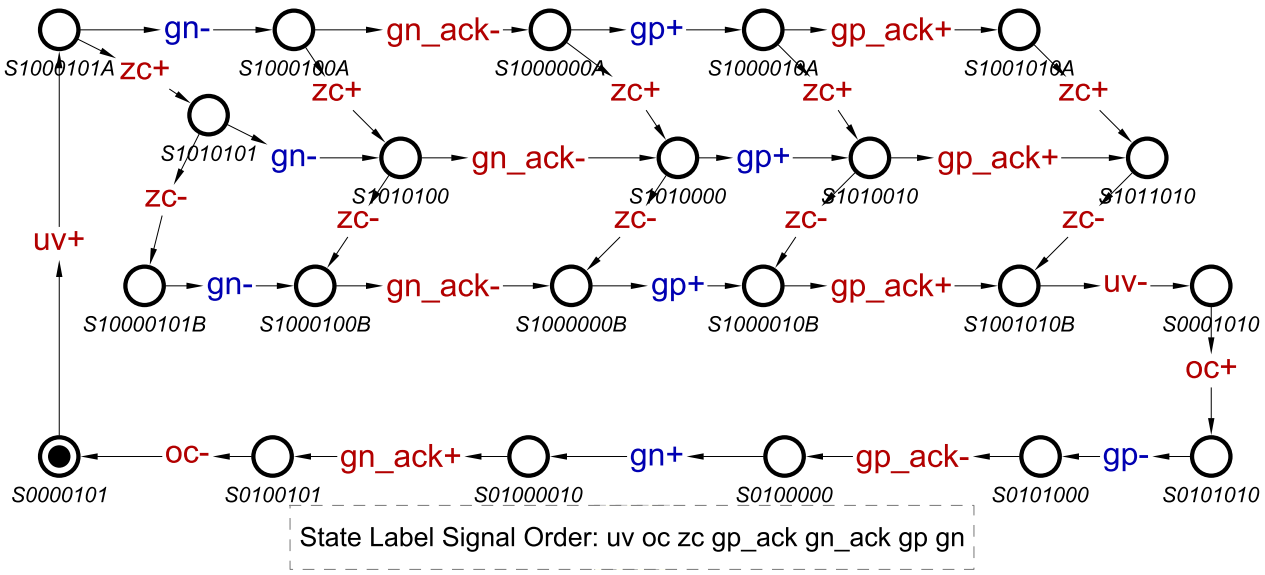

Figure 24. Full FSM model for UV before ZC

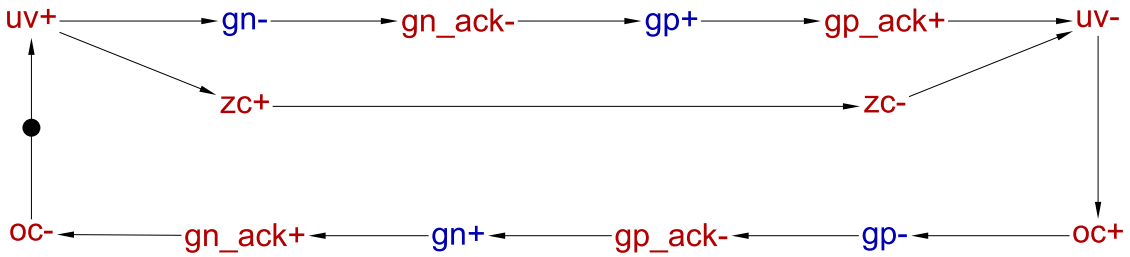

Figure 25. Full STG model for UV before ZC after resynthesis

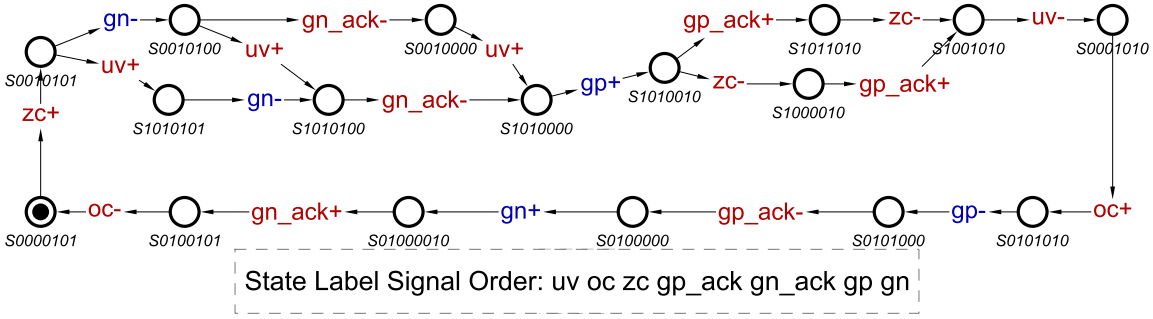

Figure 26. Full FSM model for UV after ZC

Figure 29 is the full STG for this system. As with the FSM you can see the three separate scenarios with the under voltage branches, and that there is only one branch for over current, again due to its similarity between scenarios, it has been combined. The combination process has added two places, which allow one token each. The left most place contains a token, as this is an initial state, and it allows only one of the three under voltage scenarios to be active at any time. No CSC conflicts can be seen in this model, which helps with understanding it when viewing the model.

With these models completed, and functioning correctly we can now move on to the final steps.

VII. Synthesis and Circuit

The two models can now be synthesized to find a complex gate implementation can be acquired which will be used in the actual device. Synthesis is the process of breaking down a model into working regions, and from this, calculating the logic necessary so the input signals cause the correct output signals[5]. Various software tools can be used to synthesize the circuit, and for the models in this example, I chose to use Petrify, as both model types, Finite State Machines and Signal Transition Graphs, can be implemented in the format (*.g files) used with Petrify, and a gate implementation can be found.

With both of the models used in this system, the same circuit

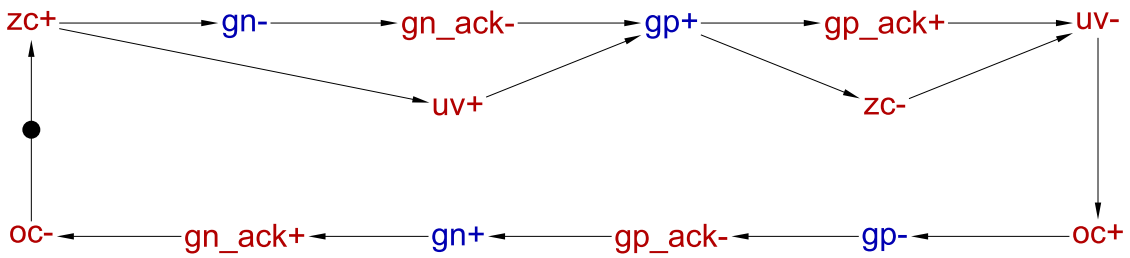

Figure 27. Full STG model for UV after ZC after resynthesis

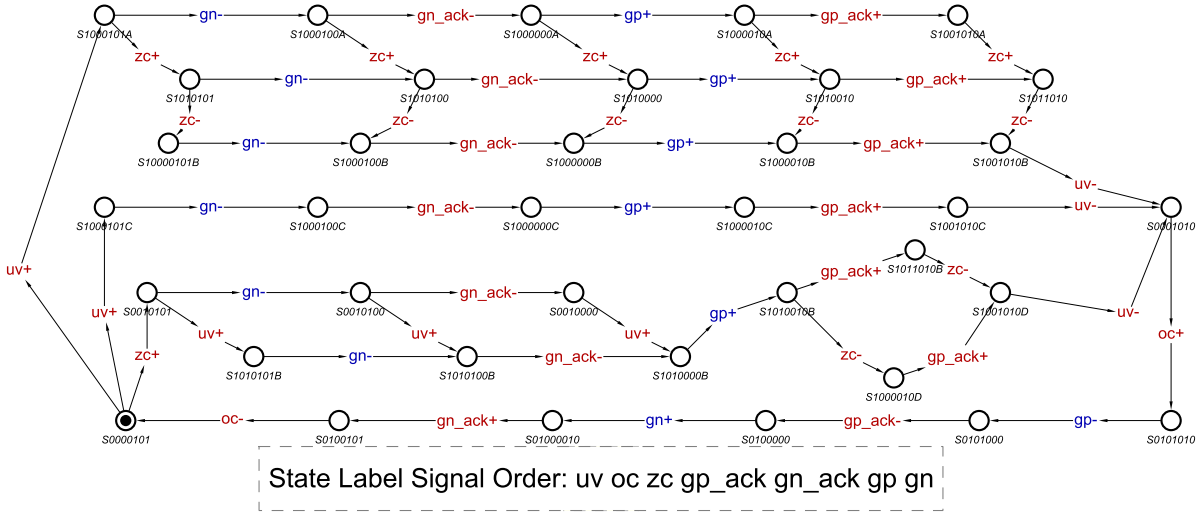

Figure 28. Full system FSM model

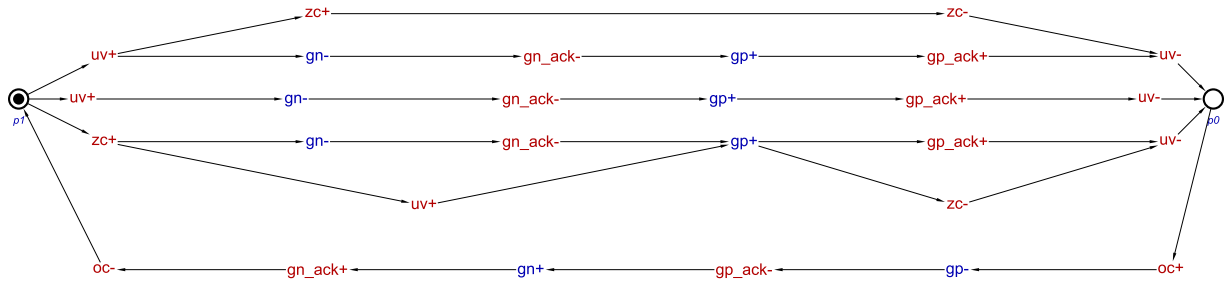

Figure 29. Full system STG model

implementation is found using Petrify. It includes the gates necessary to use the inputs uv, oc, zc, gp_ack and gn_ack, and output gp and gn.

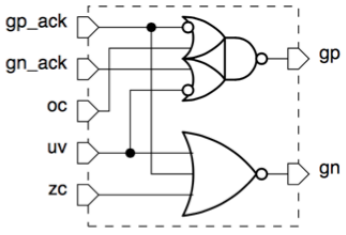

Figure 30. Complex gate implementation of both FSM and STG models [10]

VIII. Conclusions & Future Work

It has been shown that it is possible to use smaller fragments describing signal requirements of a system, which are modeled themselves, in order to create models for scenarios of the system, and from this, create a full model which can be synthesized for a circuit implementation to be used in a device. This could make developing models for a larger system less complicated, by reusing fragments, and ensuring that certain protocols are present throughout a system, which again can help to eliminate errors and consequently, design time.

While the methods described to automate some processes with FSMs do not yet exist, the methods are implementable. And while the process will work with FSMs, they are still somewhat complex when concurrency is involved. STGs would be the preferred method of composing a system from fragments with this example, but certain examples may be

better suited to use FSMs, and this method could be applied to various other modeling systems.

A third modeling method exists, knows as Conditional Partial Order Graphs (CPOG) [8], [6]. CPOGs have methods of composition built into Workcraft [9], and in the future I plan to use CPOGs, as well as FSMs and STGs, in order to compare the three methods and their usage when designing models from fragments.

REFERENCES

[1] Arseniy Alekseyev. Compositional approach to design of digital circuits. Technical report, School of Electrical and Electronic Engineering, Newcastle University, 2014.

[2] Arseniy Alekseyev, Victor Khomenko, Andrey Mokhov, Dominic Wist, and Alex Yakovlev. Improved parallel composition of labelled petri nets. In *Application of Concurrency to System Design (ACSD), 2011 11th International Conference on*, pages 131–140. IEEE, 2011.

[3] Arseniy Alekseyev, Ivan Poliakov, Victor Khomenko, and Alex Yakovlev. Optimisation of balsa control path using stg resynthesis.

[4] Michael A. Arbib. *Theories of Abstract Automata (Automatic Computation)*.

[5] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. *Logic Synthesis for Asynchronous Controllers and Interfaces*. Springer, 2002.

[6] Andrey Mokhov. *Conditional Partial Order Graphs*. PhD thesis, PhD thesis, Newcastle University, 2010.

[7] Andrey Mokhov, Maxim Rykunov, Danil Sokolov, and Alex Yakovlev. Design of processors with reconfigurable microarchitecture. *Journal of Low Power Electronics and Applications*, 4(1):26–43, 2014.

[8] Andrey Mokhov and Alex Yakovlev. Conditional partial order graphs: Model, synthesis, and application. *IEEE Transactions on Computers*, 59(11):1480–1493, 2010.

[9] Ivan Poliakov, Danil Sokolov, and Andrey Mokhov. Workcraft: A static data flow structure editing, visualisation and analysis tool. *Petri Nets and Other Models of Concurrency–ICATPN 2007*, pages 505–514, 2007.

[10] Danil Sokolov, Andrey Mokhov, Alex Yakovlev, and David Lloyd. Towards asynchronous power management. In *2014 IEEE Faible Tension Faible Consommation (FTFC)*, pages 1–4. IEEE, 2014.

[11] Fei Xia, Andrey Mokhov, Yu Zhou, Yuan Chen, Isi Mitrani, Delong Shang, Danil Sokolov, and Alex Yakovlev. Towards power-elastic systems through concurrency management. *Computers & Digital Techniques, IET*, 6(1):33–42, 2012.

[12] Alexandre V Yakovlev, Albert M Koelmans, and Luciano Lavagno. High-level modeling and design of asynchronous interface logic. *IEEE Design & Test of Computers*, 12(1):32–40, 1995.