Project 3

NuSMV

ECE 582

Abram Fouts

Mike Tian

February 17th, 2021

Table of Contents

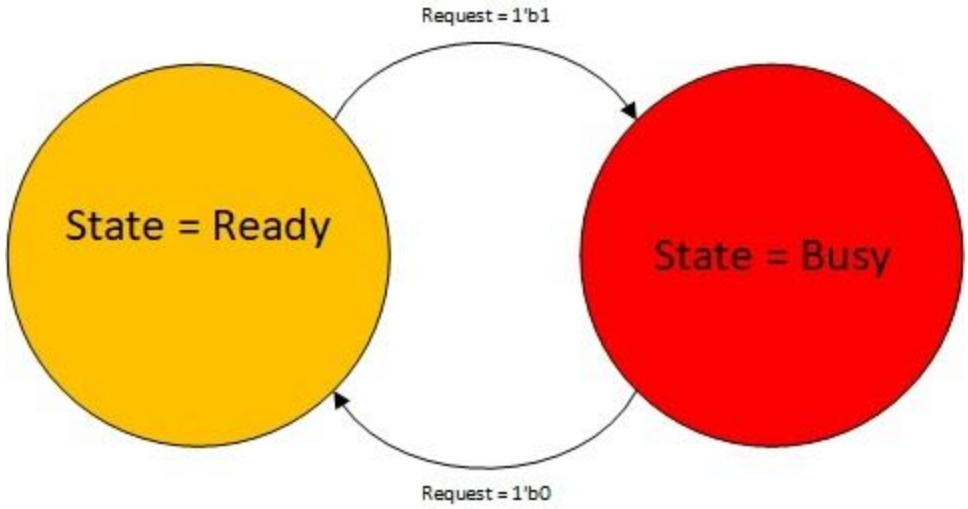
Problem 1:

```
$ ./nusmv -int
*** This is NuSMV 2.5.4 (compiled on Fri Oct 28 14:15:02 UTC 2011)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <nusmv-users@fbk.eu>

*** Copyright (c) 2010, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat
*** Copyright (c) 2003-2005, Niklas Een, Niklas Sorensson

WARNING *** This version of NuSMV is linked to the zchaff SAT         ***
WARNING *** solver (see http://www.princeton.edu/~chaff/zchaff.html). ***
WARNING *** Zchaff is used in Bounded Model Checking when the         ***
WARNING *** system variable "sat_solver" is set to "zchaff".          ***
WARNING *** Notice that zchaff is for non-commercial purposes only.   ***
WARNING *** NO COMMERCIAL USE OF ZCHAFF IS ALLOWED WITHOUT WRITTEN    ***
WARNING *** PERMISSION FROM PRINCETON UNIVERSITY.                     ***
WARNING *** Please contact Sharad Malik (malik@ee.princeton.edu)      ***
WARNING *** for details.                                              ***
```

```
NuSMV > help
add_property                          alias
bmc_inc_simulate                      bmc_pick_state
bmc_setup                             bmc_simulate
bmc_simulate_check_feasible_constraints
build_boolean_model                   build_flat_model
build_model                           check_compute
check_ctlspec                         check_fsm
check_invar                           check_invar_bmc
check_invar_bmc_inc                   check_ltlspec
check_ltlspec_bmc                     check_ltlspec_bmc_inc
check_ltlspec_bmc_onepb               check_ltlspec_sbmc
check_ltlspec_sbmc_inc                check_property
check_pslspec                         check_spec
clean_sexp2bdd_cache                  compass_gen_sigref
compute                               compute_reachable
dump_expr                             dump_fsm
dynamic_var_ordering                  echo
encode_variables                      execute_partial_traces
execute_traces                        flatten_hierarchy
gen_invar_bmc                         gen_ltlspec_bmc
gen_ltlspec_bmc_onepb                 gen_ltlspec_sbmc
get_internal_status                   go
go_bmc                                goto_state
help                                  history
hrc_dump_model                        hrc_write_model
pick_state                            print_bdd_stats
print_clusterinfo                     print_current_state
print_fair_states                     print_fair_transitions
print_formula                         print_fsm_stats
print_iwls95options                   print_reachable_states
print_usage                           process_model
quit                                  read_model
read_trace                            reset
set                                   set_bdd_parameters
show_dependencies                     show_plugins
show_property                         show_traces
show_vars                             simulate
source                                time
unalias                               unset
usage                                 which
write_boolean_model                   write_coi_model
write_flat_model                      write_flat_model_udg
write_order
NuSMV > read_model -h
usage: read_model [-h] [-i <file>]
   -h           Prints the command usage.
   -i <file>    Reads the model from the specified <file>.
NuSMV > |
```

Problem 2

1. **State Diagram**



2. **3 more CTL Properties checked against the model**

AF (request -> AX state = busy)

```
NuSMV > check_ctlspec -p "AF (request -> AX state = busy)"
-- specification AF (request -> AX state = busy)  is true
```

AX (state = busy -> AF request)

```
NuSMV > check_ctlspec -p "AX ((state = busy) -> AF request)"
-- specification AX (state = busy -> AF request)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  request = FALSE
  state = ready
-- Loop starts here
-> State: 1.2 <-
  state = busy
-> State: 1.3 <-
```

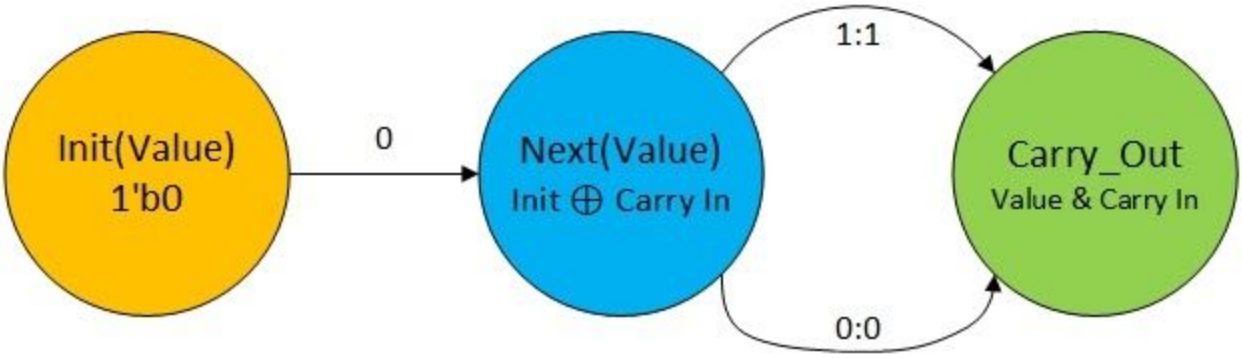
AG ((state = ready) -> AX request)

```
NuSMV > check_ctlspec -p "AG ((state = ready) -> AX request)"
-- specification AG (state = ready -> AX request)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-- Loop starts here
-> State: 2.1 <-
  request = FALSE
  state = ready
-> State: 2.2 <-
```

Problem 3

State Transition Diagram

1. Counter cell consists of 1 boolean variable, always initialized to 0. Then is *XOR*'d by the input on the module (defined below). The carry out value is assigned by *AND*ing the result of the previous *XOR* and the input to the module.



This state transition diagram for the counter cell code. This code is then recursively called 3 times using the counter cell module. The input of bit 0 is always *true*, and the output of bit 0 is in the input of bit 1, and the output of bit 1 is used as an input for bit 3.

3 CTL Models

Come up with 3 more CTL property and check against above model

EF(bit1.value)

AF(bit1.value)

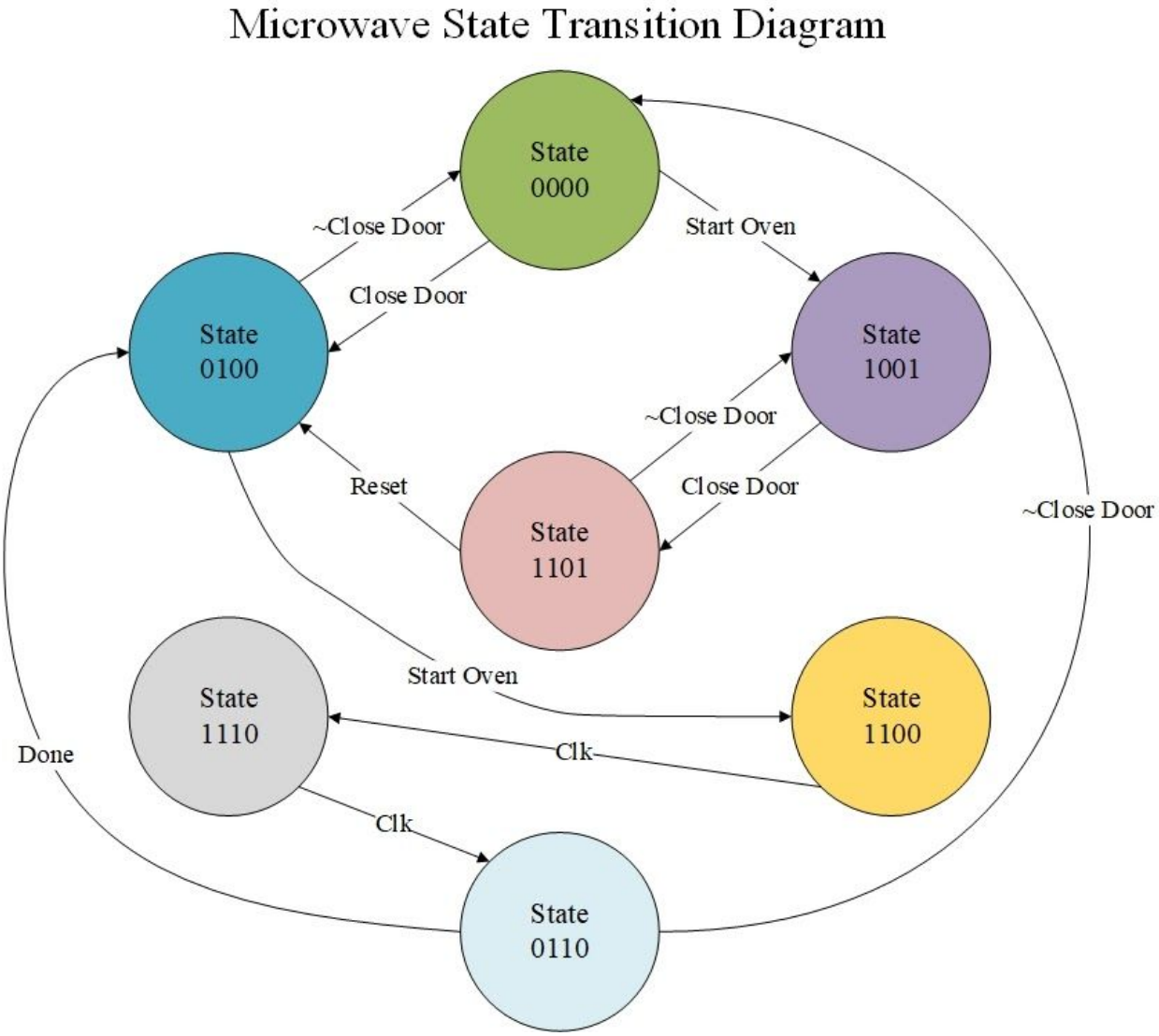
AX(bit1.value)

```
NuSMV > check_ctlspec -p "EF(bit1.value)"
-- specification EF bit1.value  is true
NuSMV > check_ctlspec -p "AF(bit1.value)"
-- specification AF bit1.value  is true
NuSMV > check_ctlspec -p "AX(bit1.value)"
-- specification AX bit1.value  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 9.1 <-
  bit0.value = FALSE
  bit1.value = FALSE
  bit2.value = FALSE
  bit0.carry_out = FALSE
  bit1.carry_out = FALSE
  bit2.carry_out = FALSE
-> State: 9.2 <-
  bit0.value = TRUE
  bit0.carry_out = TRUE
NuSMV >
```

Problem 4

State Transition Diagram

State transition diagram for the microwave SystemVerilog code is shown below,



Five CTL properties in english and CTL format with results.

#1

For all states the next state is the initial state (0000)

$AX(0000)$

```
# Top level modules:
#       microwavetb
# End time: 14:38:50 on Mar 03,2021, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
vsim work.microwavetb
# vsim
# Start time: 14:38:51 on Mar 03,2021
# Loading sv_std.std
# Loading work.microwavetb
# Loading work.microwave
run
# States= xxxx
# Asserting sys_reset on state: 0000
# States= 0000
# Test case:           0
#
# Asserting sys_reset on state: 1001
# States= 1001
# States= 0000
# Test case:           1
#
# States= 1001
# Asserting sys_reset on state: 1101
# States= 1101
# States= 0000
# Test case:           2
#
# States= 1001
# States= 1101
# Asserting sys_reset on state: 0100
# States= 0100
# States= 0000
# Test case:           3
#
# States= 1001
# States= 1101
# States= 0100
# Asserting sys_reset on state: 1100
# States= 1100
# States= 0000
# Test case:           4
#
# States= 1001
# States= 1101
# States= 0100
```

```
# States= 1100
# Asserting sys_reset on state: 1110
# States= 1110
# States= 0000
# Test case:           5
#
# States= 1001
# States= 1101
# States= 0100
# States= 1100
# States= 1110
# Asserting sys_reset on state: 0110
# States= 0110
# States= 0000
# Test case:           6
```

Here it proves that on every state of the #1 test bench that EVERY branch has a next state branch that goes to the initial state (0000) from system reset. This turns out to be true because of the system reset. This is also represented by AG(~doorClosed & ~start &~heat &~error).

#2

There exists a next state where the state is 0100

$$EX(0100)$$

```
# Top level modules:
#       microwavetb
# End time: 15:11:58 on Mar 03,2021, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
vsim work.microwavetb
# vsim
# Start time: 15:11:59 on Mar 03,2021
# Loading sv_std.std
# Loading work.microwavetb
# Loading work.microwave
run
# States= xxxx
# States= 0000
# States= 0100
# States= 1100
# States= 1110
# States= 0110
# States= 0000
```

There exists a next state where the next state is 0100, which is a necessary state to use the microwave, as proved above. From 0000 the next state can be 0100, and it is achievable from the other states as well.

#3

There exists states where all next states have no heat

$EG(\boldsymbol{NoHeat})$

```
# Top level modules:
#       microwavetb
# End time: 15:35:17 on Mar 03,2021, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
vsim work.microwavetb
# vsim
# Start time: 15:35:19 on Mar 03,2021
# Loading sv_std.std
# Loading work.microwavetb
# Loading work.microwave
run
# States= xxxx
# States= 0000
# States= 1001
# States= 1101
# States= 0100
# Test:           0
# States= 0000
# States= 1001
# States= 1101
# States= 0100
# Test:           1
# States= 0000
# States= 1001
# States= 1101
# States= 0100
# Test:           2
# States= 0000
# States= 1001
# States= 1101
# States= 0100
# Test:           3
```

There exists a grouping of states where there is no head, and it is proved by going through the infinite loop of 0000 -> 1001 -> 1101 -> 0100 back to 0000 to complete the cycle, so the statement is true.

#4

There exists a next state where done is true then the oven is off

$$EX(\text{done} \rightarrow (\text{heat} = \text{off}))$$

```
# Top level modules:
#       microwavetb
# End time: 15:54:31 on Mar 03,2021, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
vsim work.microwavetb
# vsim
# Start time: 15:54:32 on Mar 03,2021
# Loading sv_std.std
# Loading work.microwavetb
# Loading work.microwave
run
# States= xxxx
# States= 0000
# States= 0100
# States= 1100
# States= 1110
# States= 0110
# States= 0100
# States= 0000
# States= 0100
# Test(Done):           0
# States= 1100
# States= 1110
# States= 0110
# States= 0100
# States= 0000
# States= 0100
# Test(Done):           1
# States= 1100
# States= 1110
# States= 0110
# States= 0100
# States= 0000
# States= 0100
# Test(Done):           2
# States= 1100
# States= 1110
# States= 0110
# States= 0100
# States= 0000
# States= 0100
# Test(Done):           3
```

Above proved from four loops, showing that when done is asserted the path next path taken is to the state 0100.

#5

There exists a next state where error is true then the has been started

$$EX(\text{error} \rightarrow (\text{started} = \text{true}))$$

```
# Top level modules:
#       microwavetb
# End time: 16:01:59 on Mar 03,2021, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
vsim work.microwavetb
# vsim
# Start time: 16:02:01 on Mar 03,2021
# Loading sv_std.std
# Loading work.microwavetb
# Loading work.microwave
run
# States= xxxx
# States= 0000
# States= 1001
# States= 1101
# States= 1001
# States= 0000
# States= 1001
# States= 1101
# States= 1001
# States= 0000
# States= 1001
# States= 1101
# States= 1001
# States= 0000
# States= 1001
# States= 1101
# States= 1001
# States= 0000
# States= 1001
```

Every time when started was asserted in the initial state an error occurred, proving when error is true started has been asserted.

NuSMV CTL

#1 Results

check_ctlspec -p "AX(!start & !closed & !heat & !error)"

```
NuSMV > check_ctlspec -p "AX(!start & !closed & !heat & !error)"
-- specification AX (((!start & !closed) & !heat) & !error)  is true
```

As expected the results turned out to be correct.

#2 Results

check_ctlspec -p “EX(!start & closed & !heat & !error)”

```
NuSMV > check_ctlspec -p "EX(!start & closed & !heat & !error)"
-- specification EX (((!start & closed) & !heat) & !error)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  start = FALSE
  closed = FALSE
  reset = FALSE
  done = FALSE
  heat = FALSE
  error = FALSE
  sys_reset = FALSE
```

Unexpectedly this test was false, which I’m not too sure why because at some point there is a next state that meets this condition, 0100.

#3 Results

check_ctlspec -p “EG(!heat)”

```
NuSMV > check_ctlspec -p "EG(!heat)"
-- specification EG !heat  is true
```

As expected there is a global section that there is no heat. This is 0000, 0100, 1001, 1101.

#4 Results

check_ctlspec -p “EX(done -> (heat=FALSE))”

```
NuSMV > check_ctlspec -p "EX(done -> (heat=FALSE))"
-- specification EX (done -> heat = FALSE)  is true
```

This condition is true as well, it was created with the idea that that every time something is asserted or needs to be asserted, heat needs to be turned off and it’s true.

#5 Results

check_ctlspec -p “EX(error -> (start = TRUE))”

```
NuSMV > check_ctlspec -p "EX(error -> (start=TRUE))"
-- specification EX (error -> start = TRUE)  is true
```

The last case is also true, that if there is an error start has been asserted. This is shown in states, 1001 and 1101.

NuSMV Code

```
MODULE main
	VAR
		start : boolean;
		closed : boolean;
		reset : boolean;
		done : boolean;
		heat : boolean;
		error : boolean;
		sys_reset : boolean;

	ASSIGN
		init(error) := FALSE;
		init(heat) := FALSE;
		init(closed) := FALSE;
		init(start) := FALSE;
		init(done)	:= FALSE;
		init(reset) := FALSE;
		init(sys_reset) := FALSE;

		next(start) :=
			case
				sys_reset : FALSE;
				TRUE		: start;
			esac;

		next(closed) :=
			case
				sys_reset	: FALSE;
				TRUE		: closed;
			esac;

		next(error) :=
			case
				start & !closed : TRUE;
				closed & reset	: FALSE;
				sys_reset		: FALSE;
				TRUE			: error;
			esac;

		next(heat) :=
			case
				start & closed : TRUE;
				!closed			: FALSE;
				done & closed  : FALSE;
				sys_reset		: FALSE;
				TRUE			: heat;
			esac;
```

Conclusion

Each method has their advantages, but in my opinion the NuSMV is significantly better because you don't have to directly choose each state, and telling it what to do to each variable. SV testbenches could be better because you get to directly see where everything is going in each state and easily understand what you are writing.