EECS 219C: Formal Methods — Assignment 2

Parker Ziegler

March 21, 2022

1. Interrupt-Driven Program

a. Describing properties of the `Sys` module

We can describe the properties of the `Sys` module as follows:

1. `invariant main_ISR_mutex` — this property requires that execution of `main` and `ISR` is mutually exclusive. That is, if `main` is executing, `ISR` cannot be executing at the same time (or vice versa).
2. `property[LTL] one_step_ISR_return` — this property requires that, globally, if `ISR` has just returned then, in the next state, `ISR` will not return.
3. `property[LTL] main_after_ISR` — this property requires that, globally, if `ISR` is currently enabled to run and, in the next state, `main` is enabled to run, this implies that `ISR` has just returned.
4. `property[LTL] ISR_after_main` — this property requires that, globally, if `main` is enabled and, in the next state, `ISR` is enabled, this implies that an interrupt has occurred.

b. Interpreting counterexamples from the verifier

Running `uclid` with all properties commented out *except* for `main_after_ISR` results in the following counterexample:

```
CEX for vobj [Step #3] property main_after_ISR:safety @ IntSW.ucl, line 105
==========================================
Step #0
  mode : main_t
  M_enable : true
  I_enable : false
  return_ISR : false
  assert_intr : initial_1570_assert_intr
==========================================
==========================================
Step #1
  mode : ISR_t
  M_enable : true
  I_enable : false
  return_ISR : false
  assert_intr : false
==========================================
==========================================
Step #2
  mode : main_t
  M_enable : false
  I_enable : true
```

```
  return_ISR : false
  assert_intr : false
=================================
=================================
Step #3
  mode : main_t
  M_enable : true
  I_enable : false
  return_ISR : false
  assert_intr : false
=================================

// BMC counterexamples for step 4 and step 5 elided.

Finished execution for module: Sys.
```

In this counterexample, the violation of `main_after_ISR` occurs between step 2 and step 3 of the transition system. At step 2, we see that `I_enable` is set to *true* and `M_enable` is set to *false*, indicating that `ISR` is enabled to run. Additionally, the value of `return_ISR` is *false*, indicating that `ISR` has not yet returned. In the next step, `I_enable` is *false* and `M_enable` is *true*; this indicates that `ISR` should have completed running and `main` can safely be enabled. However, `return_ISR` is still *false*. In this instance, we have a case where `ISR` *was enabled*, but we have no indication that it ever returned before `main` was enabled. This violates the property that if `ISR` was enabled and, in the next state, `main` is enabled, then `return_ISR` *must* be *true*.

Running `uclid` with all properties commented out *except* for `ISR_after_main` results in the following counterexample:

```
CEX for vobj [Step #2] property ISR_after_main:safety @ IntSW.ucl, line 106
=================================
Step #0
  mode : main_t
  M_enable : true
  I_enable : false
  return_ISR : false
  assert_intr : false
=================================
=================================
Step #1
  mode : ISR_t
  M_enable : true
  I_enable : false
  return_ISR : false
  assert_intr : false
=================================
=================================
Step #2
  mode : main_t
  M_enable : false
  I_enable : true
  return_ISR : false
  assert_intr : false

// BMC counterexamples for steps 3, 4, and 5 elided.
```

```
Finished execution for module: Sys.
```

In this counterexample, the violation of `ISR_after_main` occurs between step 1 and step 2 of the transition system. At step 1, we see that `M_enable` is set to *true* while `I_enable` is set to *false* and we are "in" `ISR`. In step 2, `I_enable` is *true*, `M_enable` is *false*, and we are "in" `main`. This second state would indicate that an interrupt has been issued; however, `assert_intr` is *false*. This indicates that we transitioned into enabling `ISR` without an interrupt ever occurring. This violates the property that, if `main` is enabled in the current state and, in the next state, `ISR` is enabled, then `assert_intr` *must* be *true*.