Verification Techniques with Linux Examples

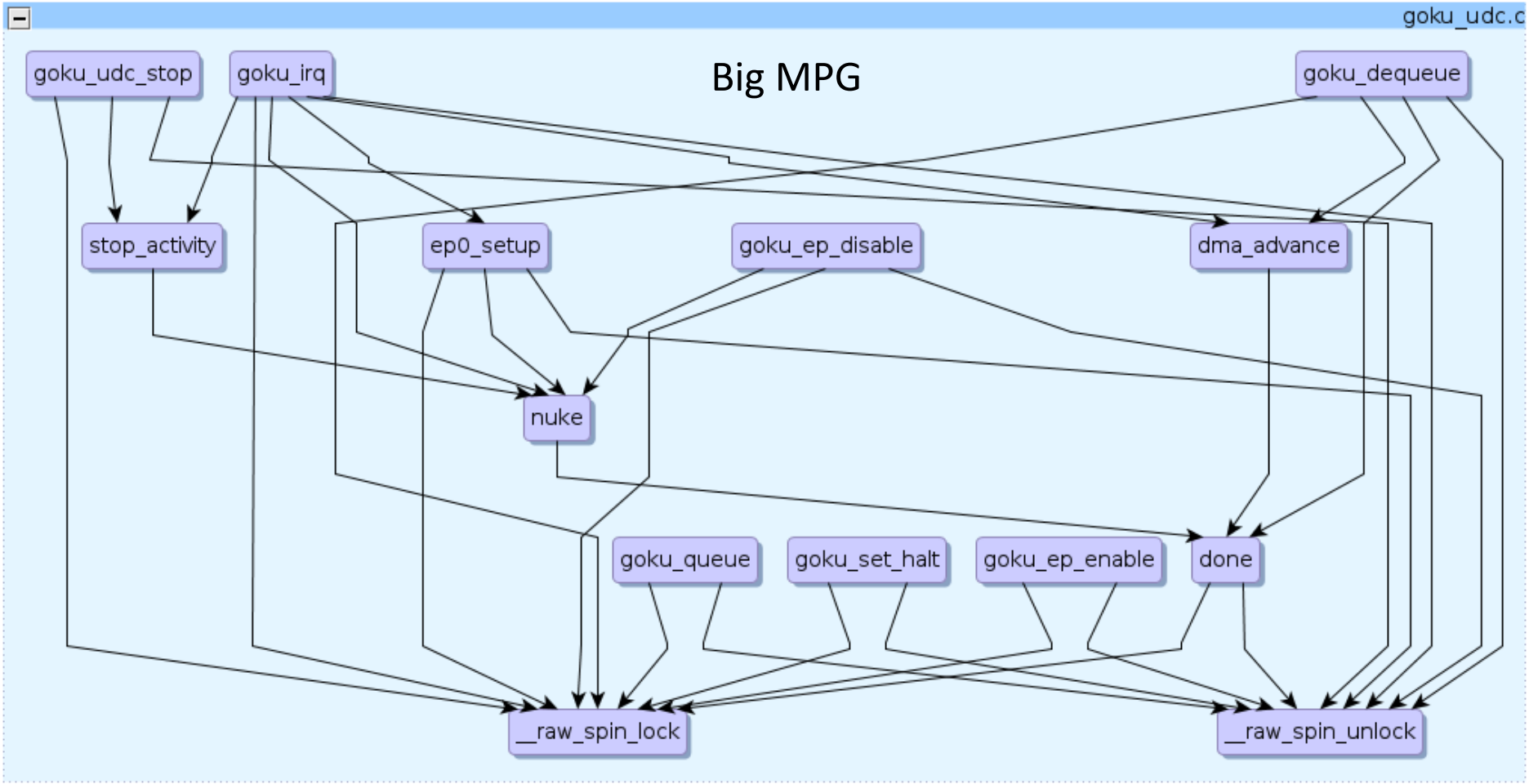
March 31, 2016

Suresh C. Kothari
Department of Electrical and Computer Engineering

Overview

These slides cover the following topics:

1. Matching Pair Graph (MPG): a call graph of functions with *direct* or *indirect* calls to LOCKS which should be verified as single collection to optimize the verification efforts.
2. A verification strategy based on direct and indirect callers of LOCK
3. CFG and *projections of EFGs* to verify a LOCK instance
4. *Encapsulation* and *reuse* of path feasibility check
5. *Modular verification* using the MPG
6. Illustrations of the above concepts
7. A complete verification of all functions in an MPG



The Big MPG shows 9 instances of LOCK calls that must be verified individually.

NOTE: There could be more than 9 instances if a function has multiple LOCK calls. It turns out be 11 LOCK instances here.

A strategy to verify the LOCK instances:

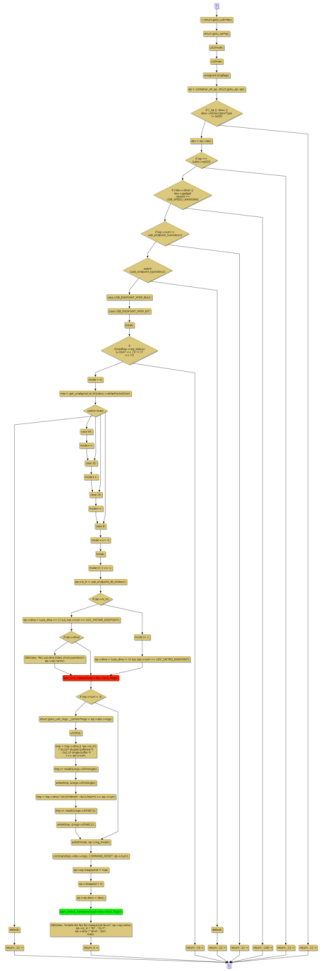
1. Categorize the functions of the MPG into two groups: (a) *direct call functions* (dcf) where the function has only *direct* calls to LOCK and UNLOCK, (b) *indirect call functions* (icf) where the function has at least one *indirect* call to LOCK or UNLOCK. The Big MPG shows 4 functions in category (a) and 8 functions in category (b).
2. Intuition: Category (a) would be easy to verify. Of these 3 (*goku_queue*, *goku_set_halt*, *goku_ep_enable*) easy to verify. The 4th case, the *done* function has the reverse order, UNLOCK followed by LOCK.
3. The UNLOCK call in *done* can only pair with a LOCK call in another function that directly or indirectly calls *done*. The same for LOCK call in *done*.
4. Among the icfs, *goku_deque*, *goku_irq*, *goku_udc_stop*, *goku_ep_disable*, and *ep0_setup* are the candidates for pairing with *done* because these functions have direct calls that can pair with the LOCK and UNLOCK calls in *done*.

Illustrations of Verifications

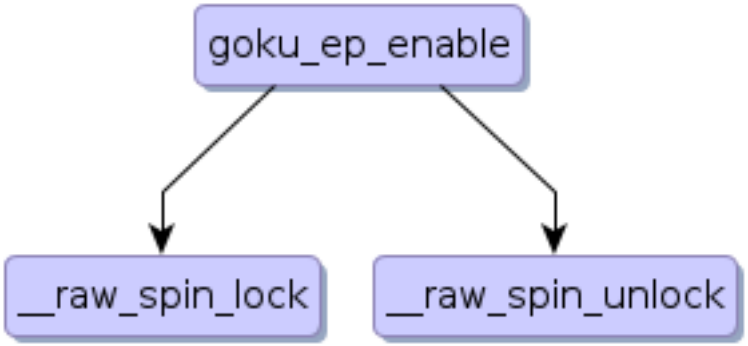
1. Illustration of easy cases: *goku_queue* and *goku_ep_enable*
2. Illustration of *goku_ep_disable*
3. Illustration of *done*
4. Illustration of *goku_irq.*

NOTE: *goku_irq* verification can be considered a more difficult case compared to *goku_deque*, *goku_udc_stop*, and *goku_ep_disable*. It is more difficult in the following way. Unlike these other functions, *goku_irq* calls *ep0_setup* which also has direct call to LOCK and UNLOCK. Thus, the added complication is to consider two pairing possibilities, either the LOCK call in *ep0_setup* or the LOCK call in *goku_irq* could pair with the UNLOCK call in *done*.

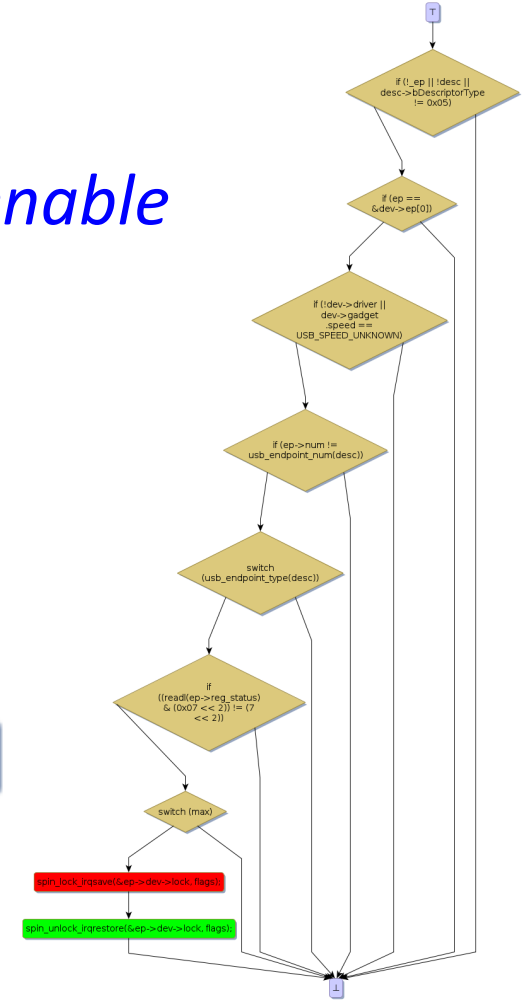
Verification of *goku_ep_enable*



CFG *goku_ep_enable*

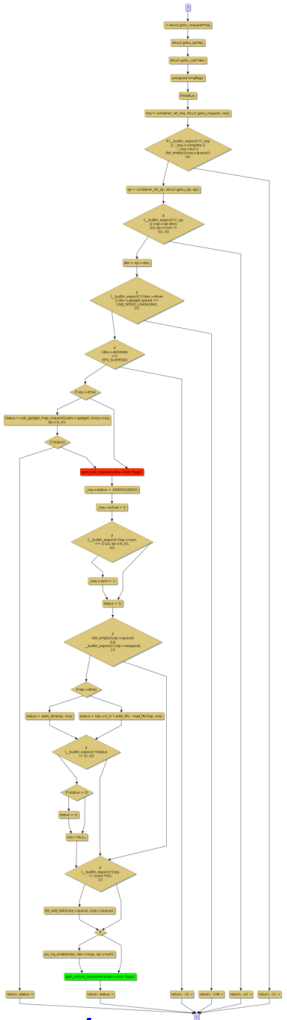


MPG for just the function *goku_ep_enable*

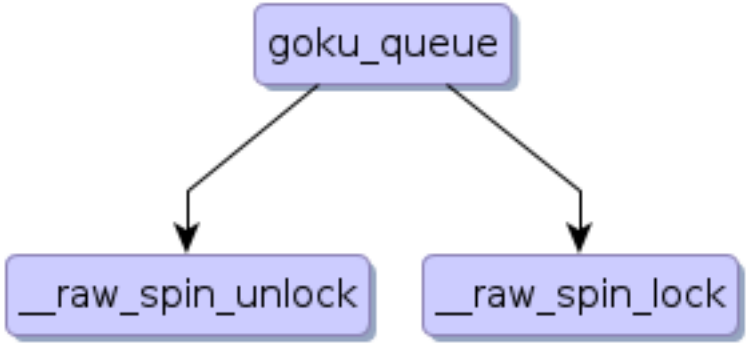


EFG *goku_ep_enable*

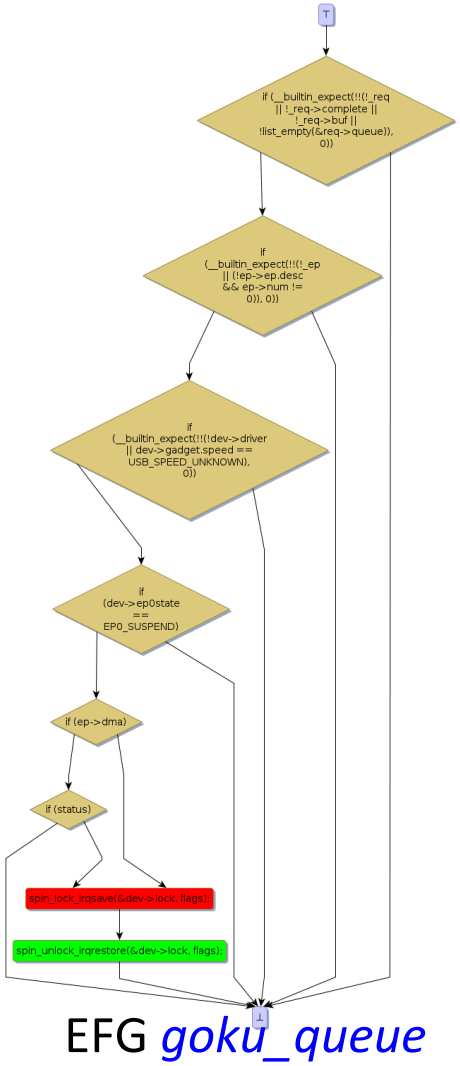
Verification of *goku_queue*



CFG *goku_queue*



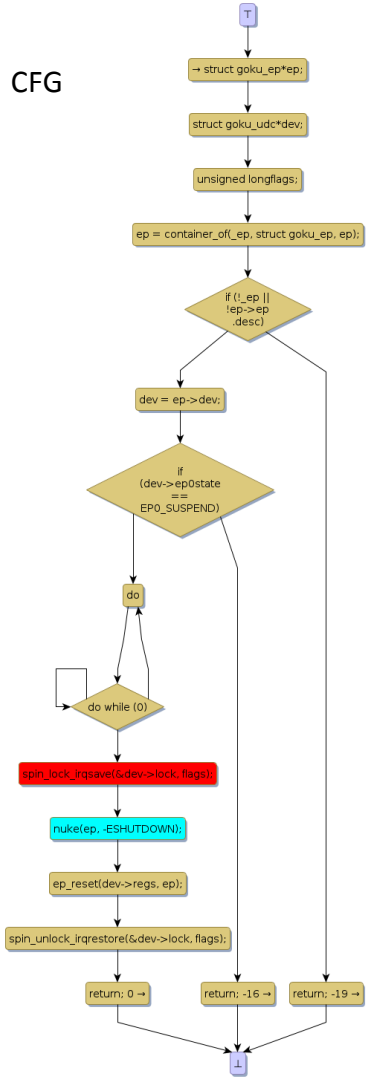
MPG for just the function
goku_queue



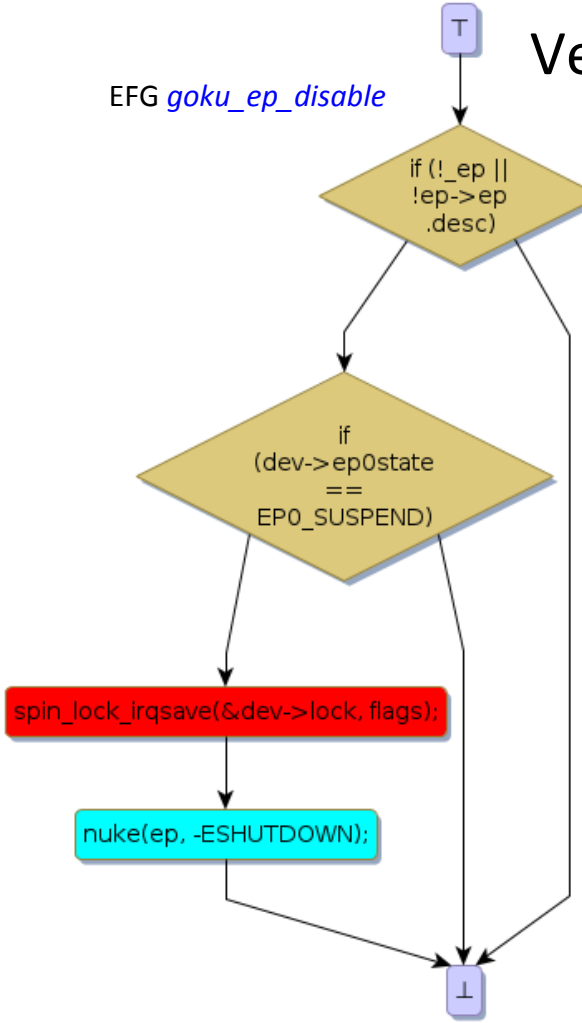
EFG *goku_queue*

Verification of *goku_ep_disable*

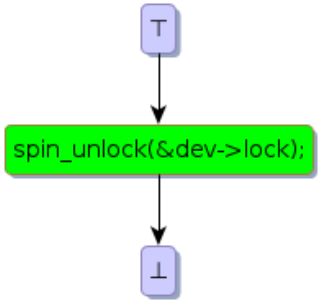
CFG

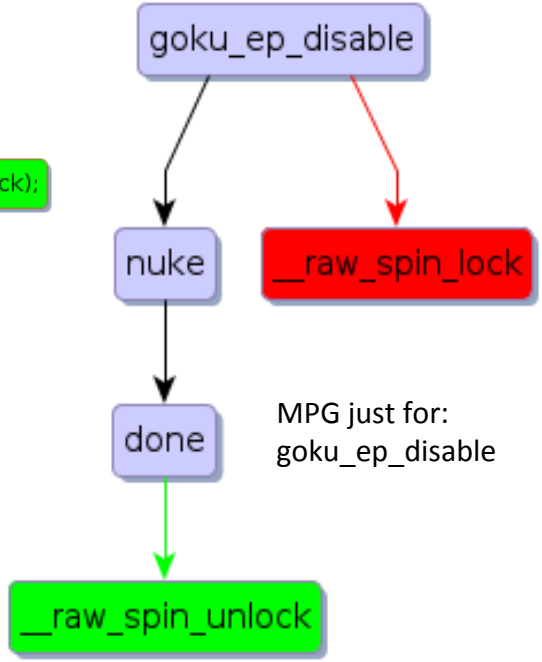


EFG *goku_ep_disable*



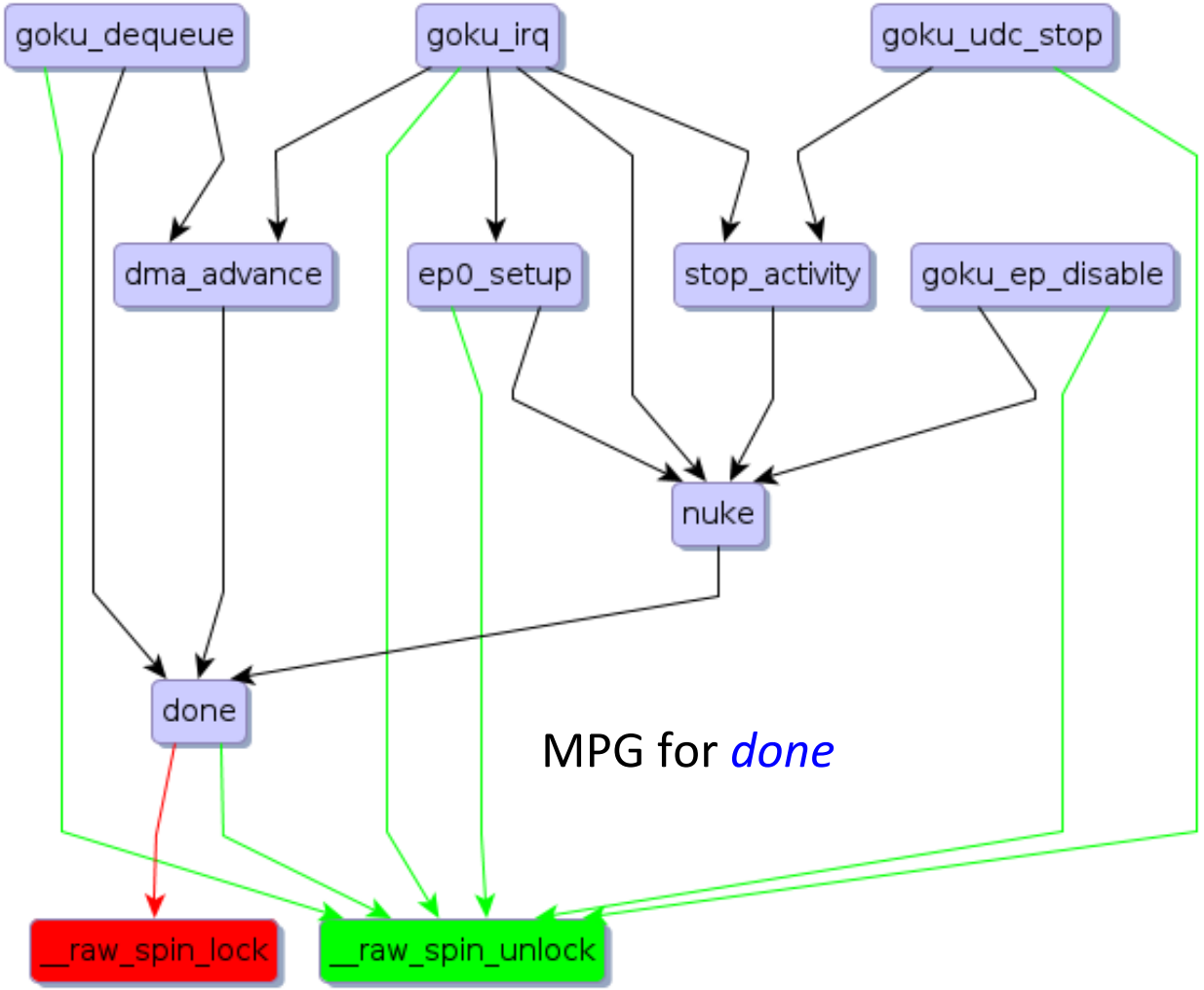
EFG *done*





MPG just for:
goku_ep_disable

The LOCK in *goku_ep_disable* pairs with the UNLOCK in *done*



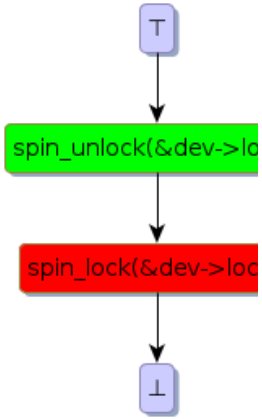
MPG for *done*

The MPG shows that the LOCK in *done* can be paired with UNLOCKs in 6 functions.

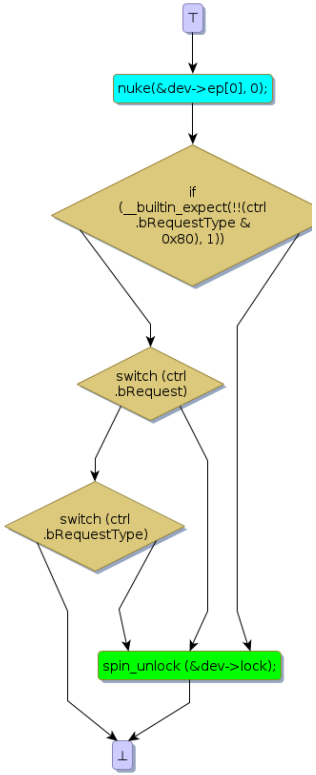
The projected EFG for *done* will show the LOCK. The projected EFGs for the six functions will show the corresponding UNLOCK in those functions.

Projected EFGs w.r.t. the LOCK in *done*

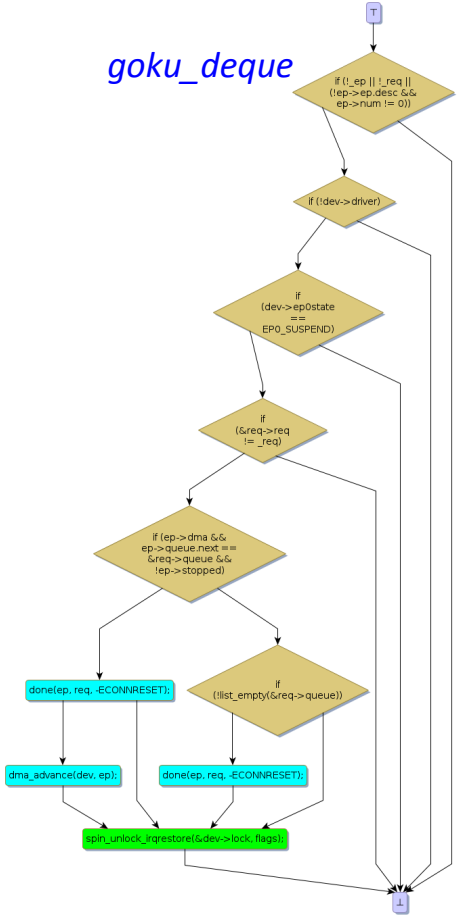
done



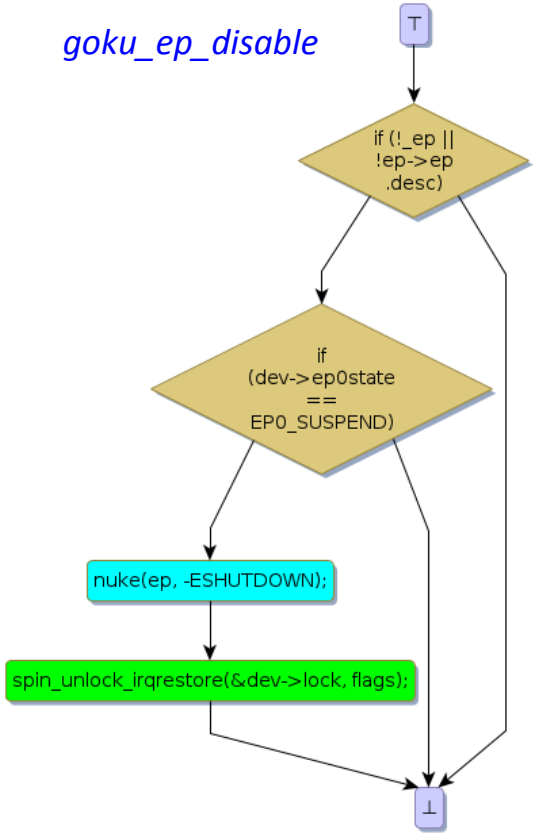
ep0_setup



goku_deque

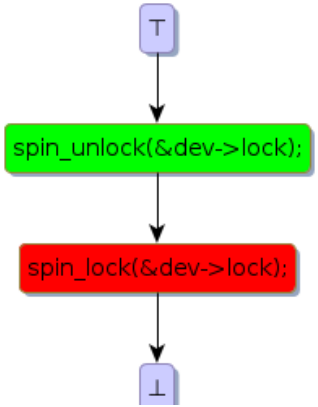


goku_ep_disable

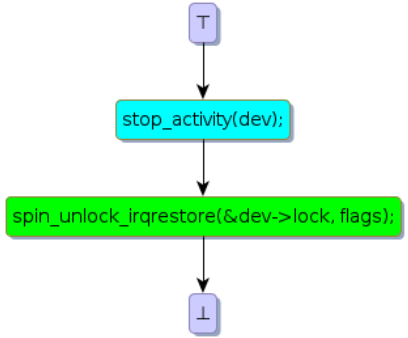


Projected EFGs w.r.t. the LOCK in *done*

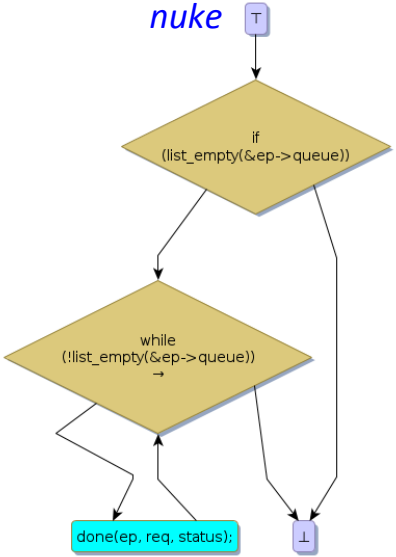
done



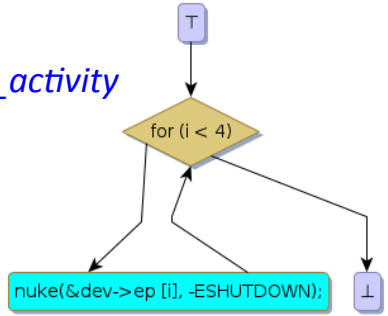
goku_udc_stop



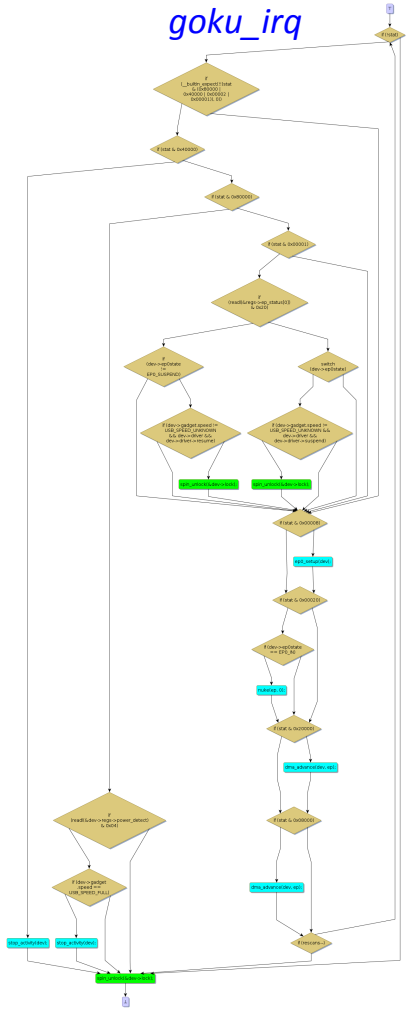
nuke



stop_activity



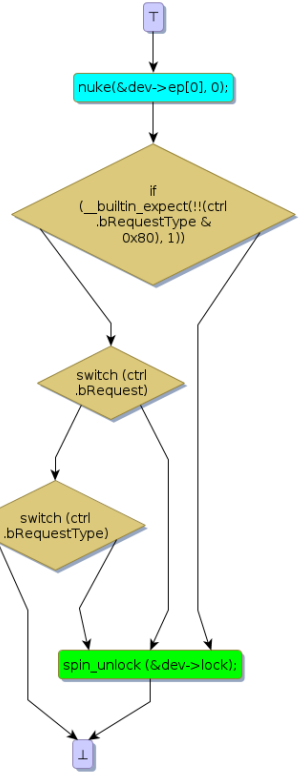
goku_irq



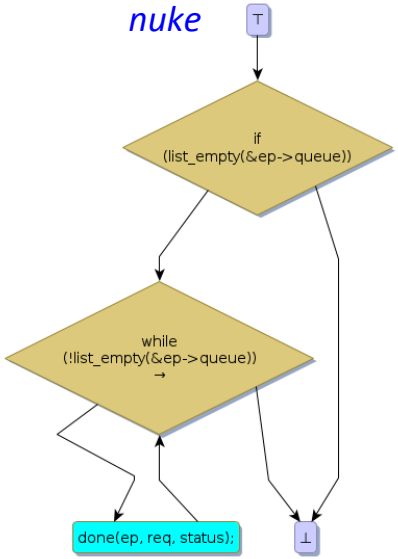
Interesting questions for verification of the LOCK in *done*

1. Why are the *nuke* and *stop_activity* relevant?
2. The *done* EFG also shows the UNLOCK, why is that relevant?
3. The EFGs for *ep0_setup* and *goku_deque* show UNLOCK only on a subset of paths. Is the LOCK not paired on the remaining paths? Is it a bug?
4. Why does *goku_irq* have an UNLOCK is followed by an UNLOCK?

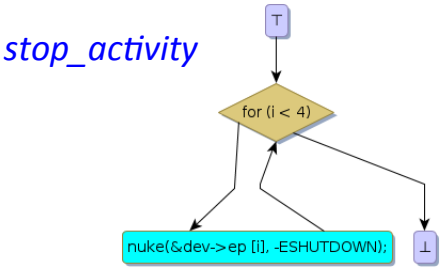
ep0_setup



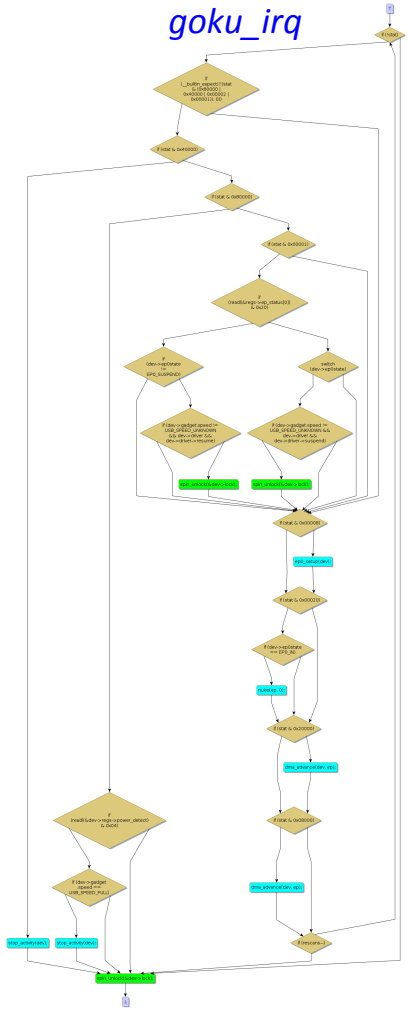
nuke



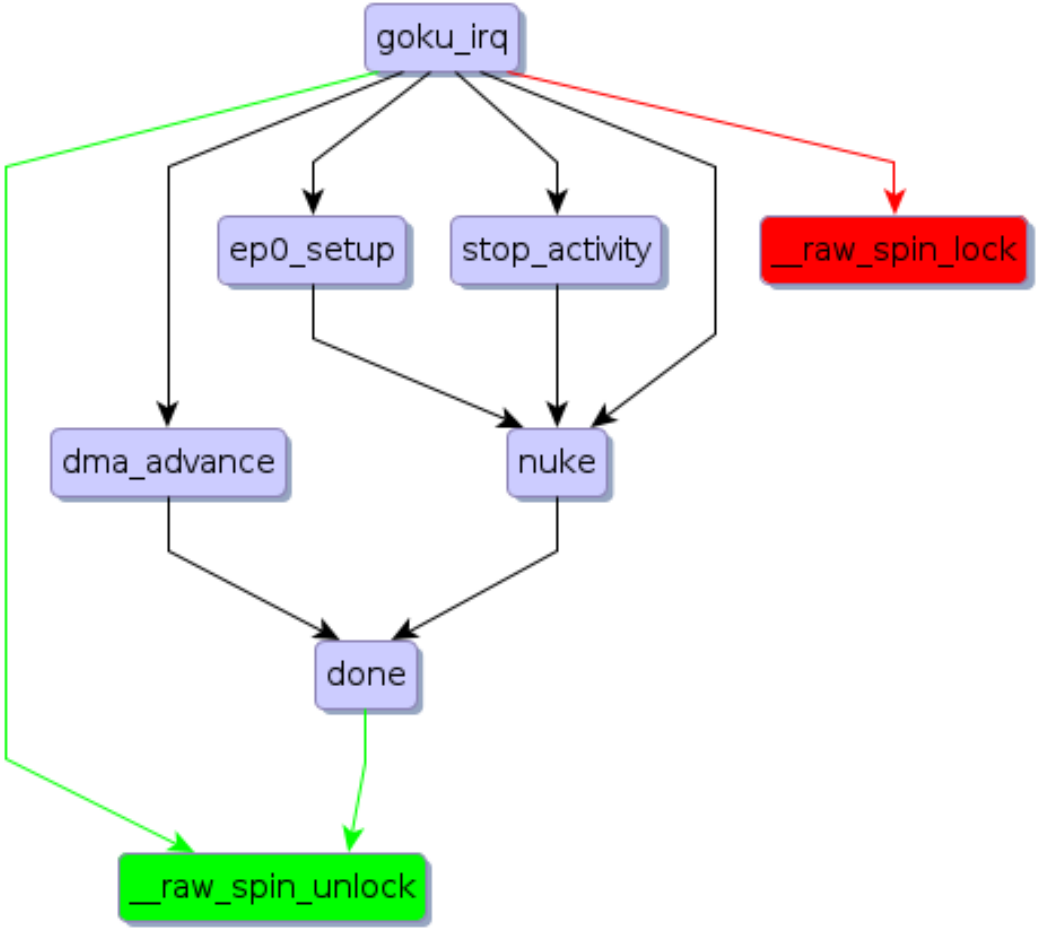
stop_activity



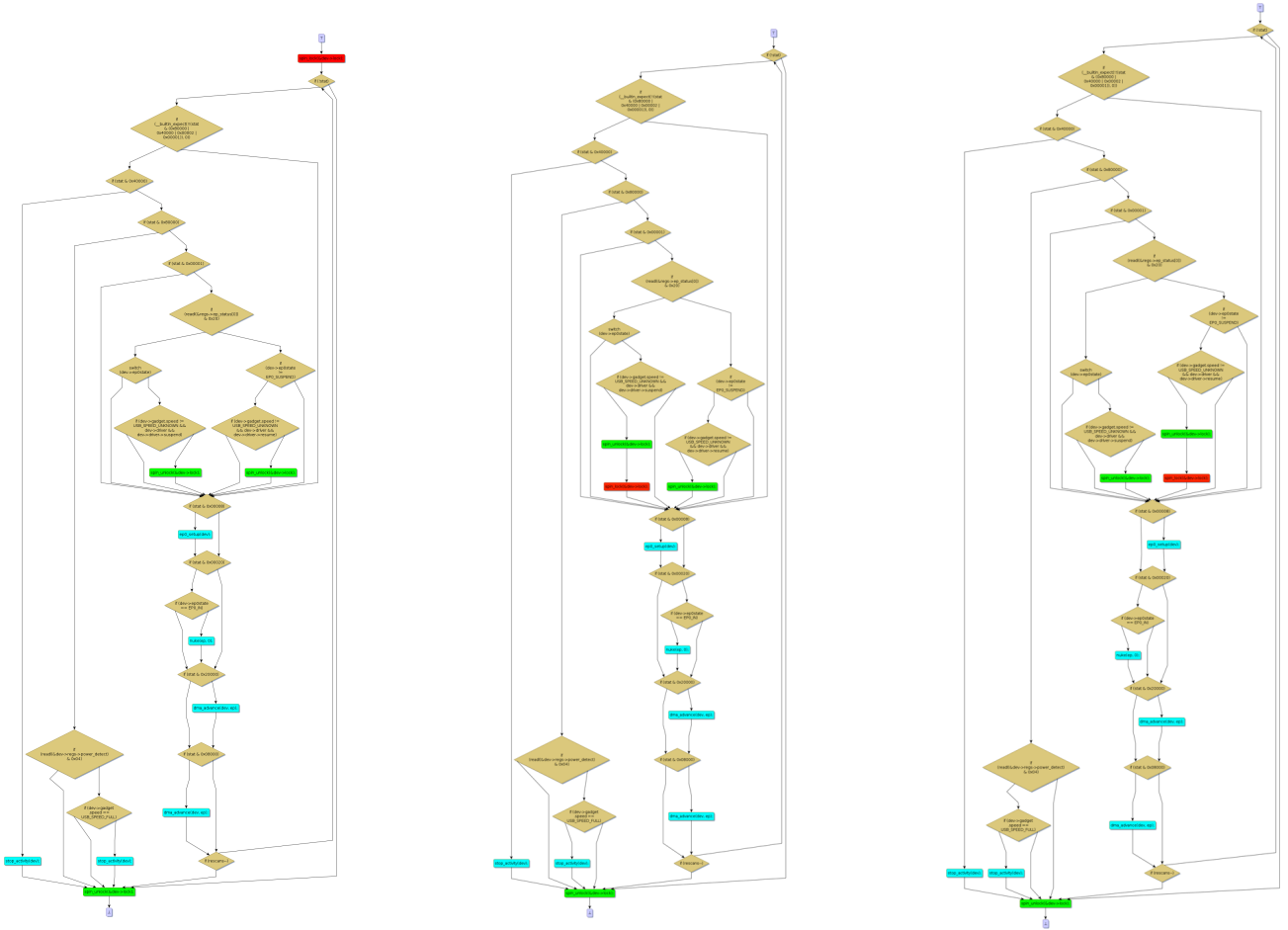
goku_irq



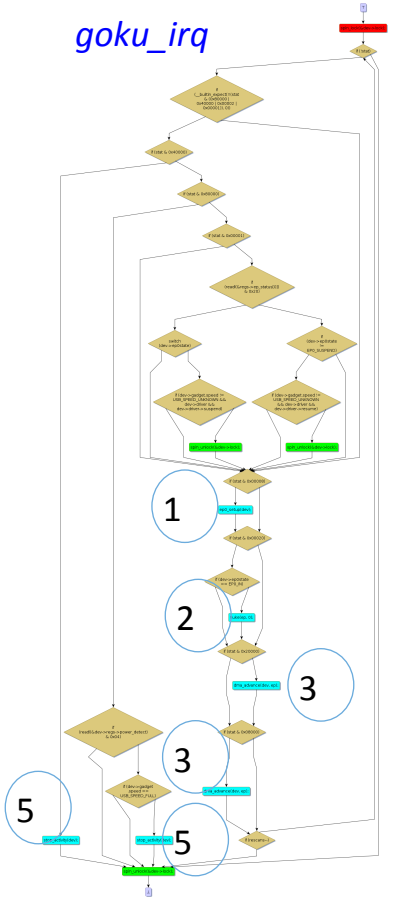
MPG for *goku_irq*

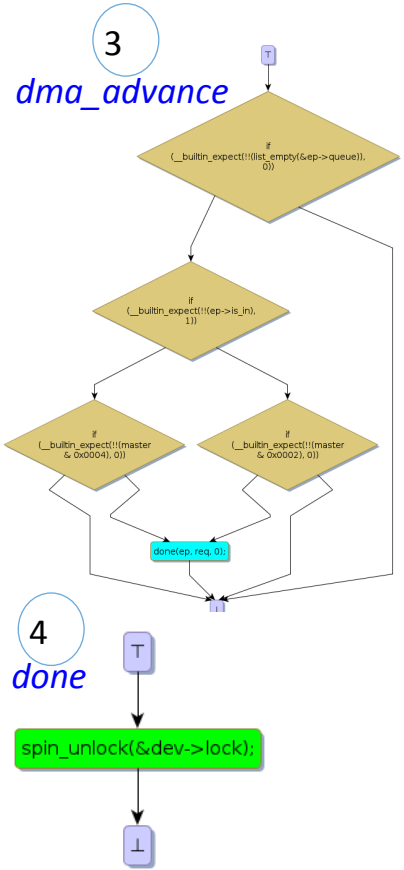


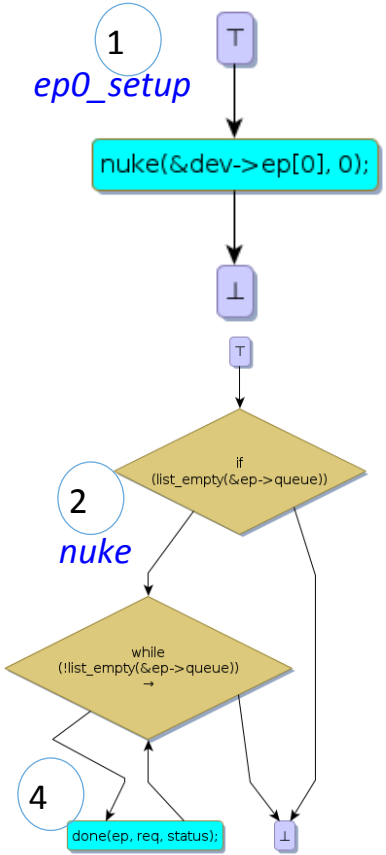
EFG for each of the three LOCKs in *goku_irq*

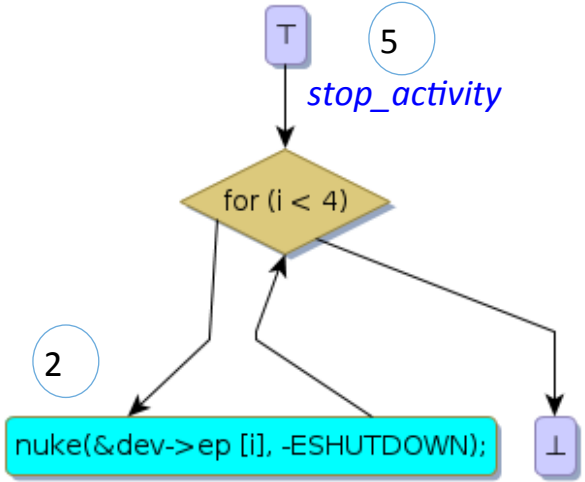


All the EFGs for the first LOCK in *goku_irq*



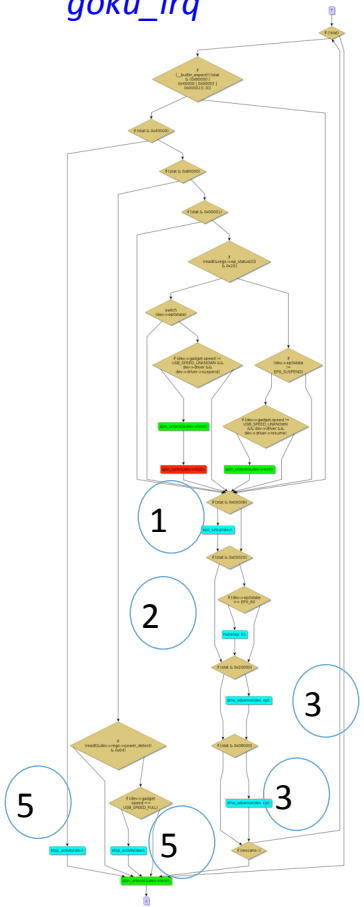




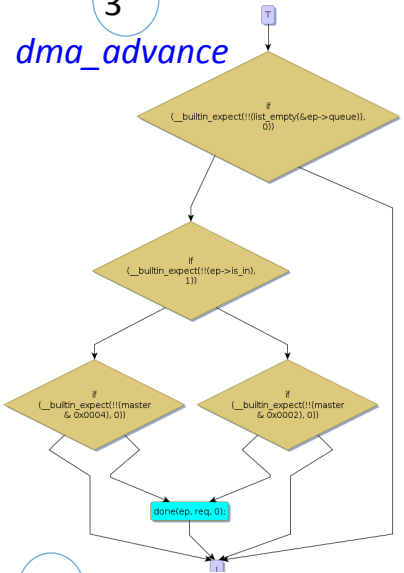


All the EFGs for the second LOCK in *goku_irq*

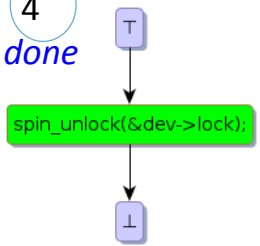
goku_irq



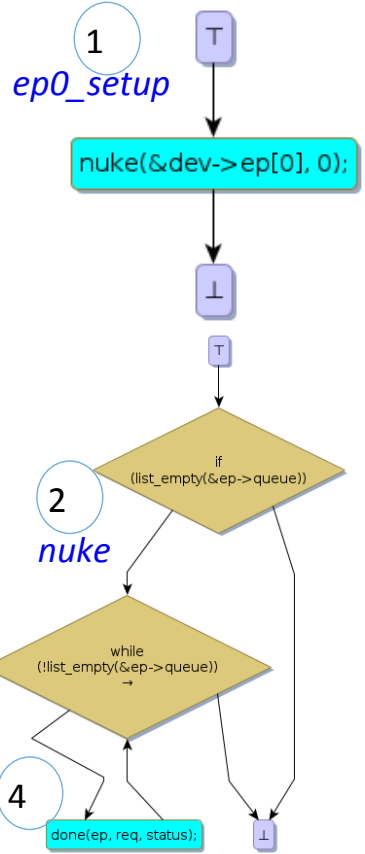
3
dma_advance



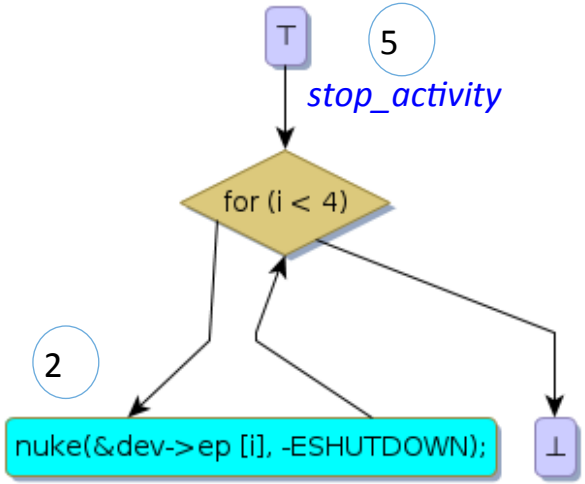
4
done



1
ep0_setup

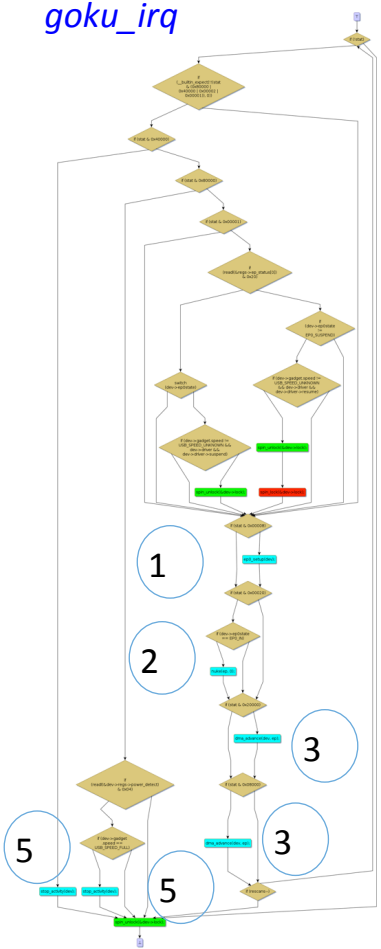


5
stop_activity

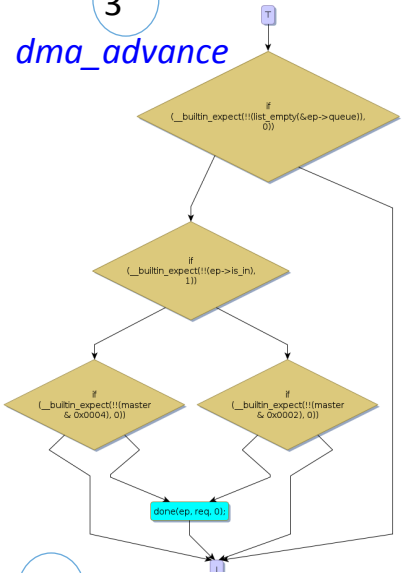


All the EFGs for the third LOCK in *goku_irq*

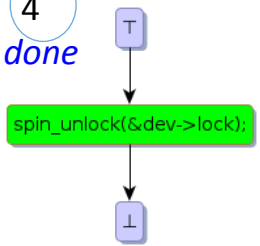
goku_irq



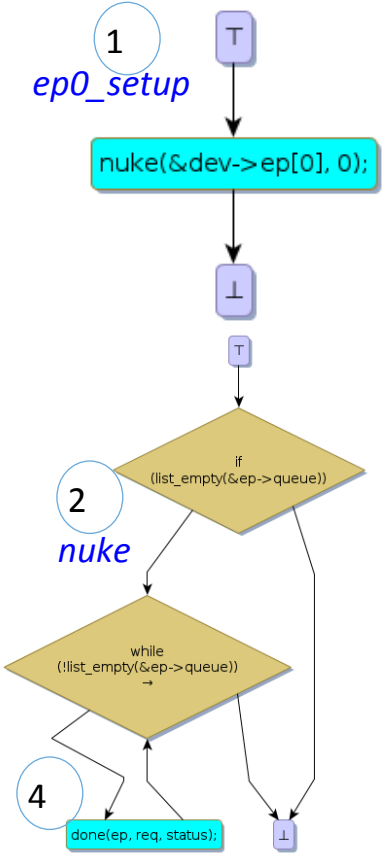
3
dma_advance



4
done

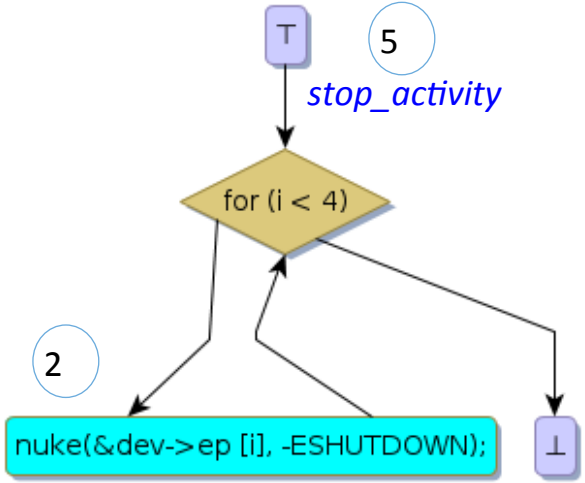


1
ep0_setup

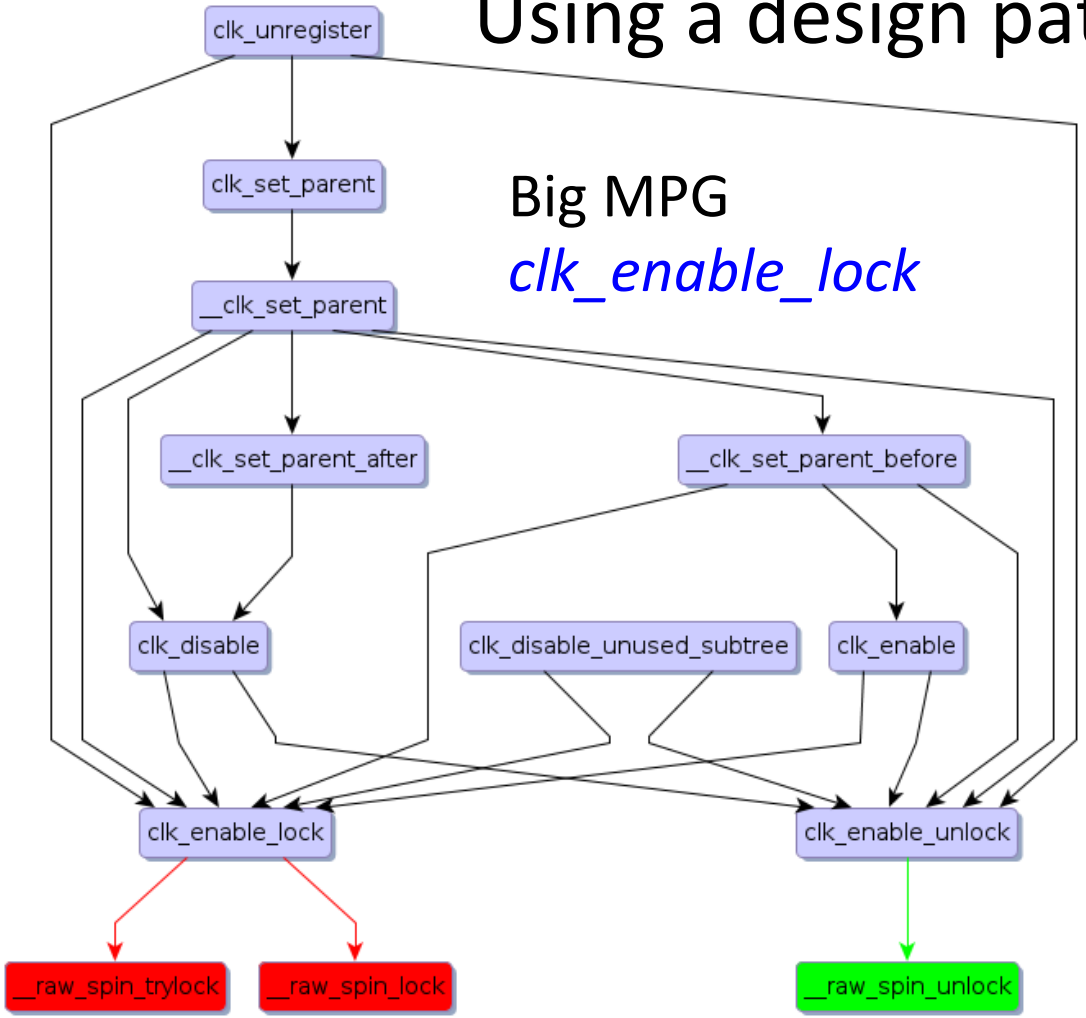


2
nuke

5
stop_activity



Using a design pattern for verification



1. *clk_enable_lock* is called from six functions. The verification has to be done for each of those functions.
2. *clk_enable_lock* has two calls to LOCK. The verification has to be done for each of those calls.

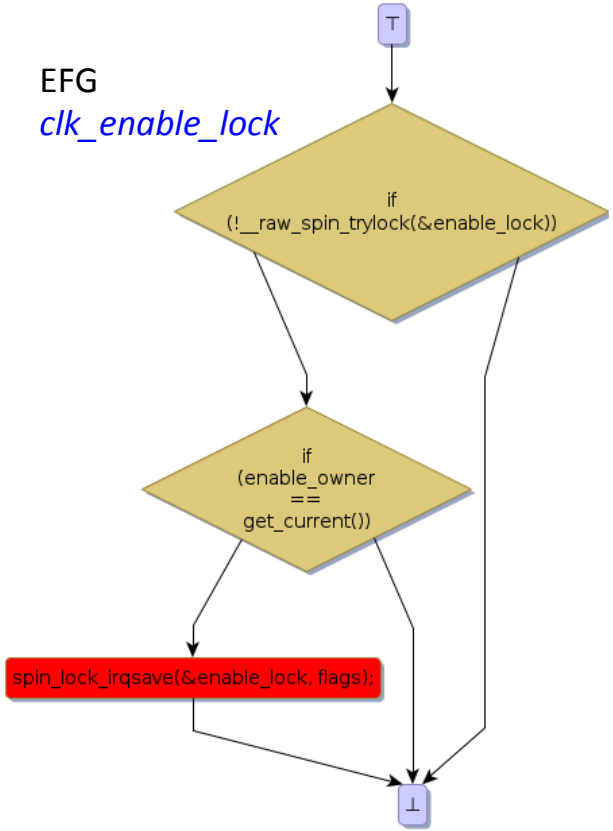
Thus, it appears that we have to do 12 verifications. We will show there is a nice design pattern because of which the verifications for the two locks do not have to be done separately.

The verification requires a *path feasibility* check. Because of the design pattern, the path feasibility check is done once and reused multiple times.

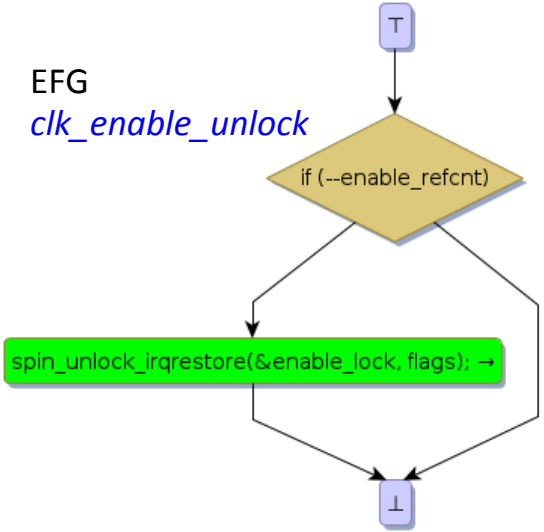
The verification effort is reduced significantly by using the design pattern.

Need for the path feasibility check

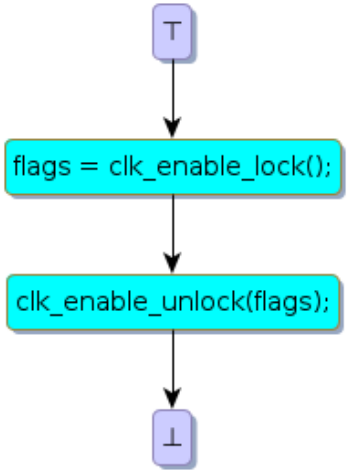
EFG
clk_enable_lock



EFG
clk_enable_unlock

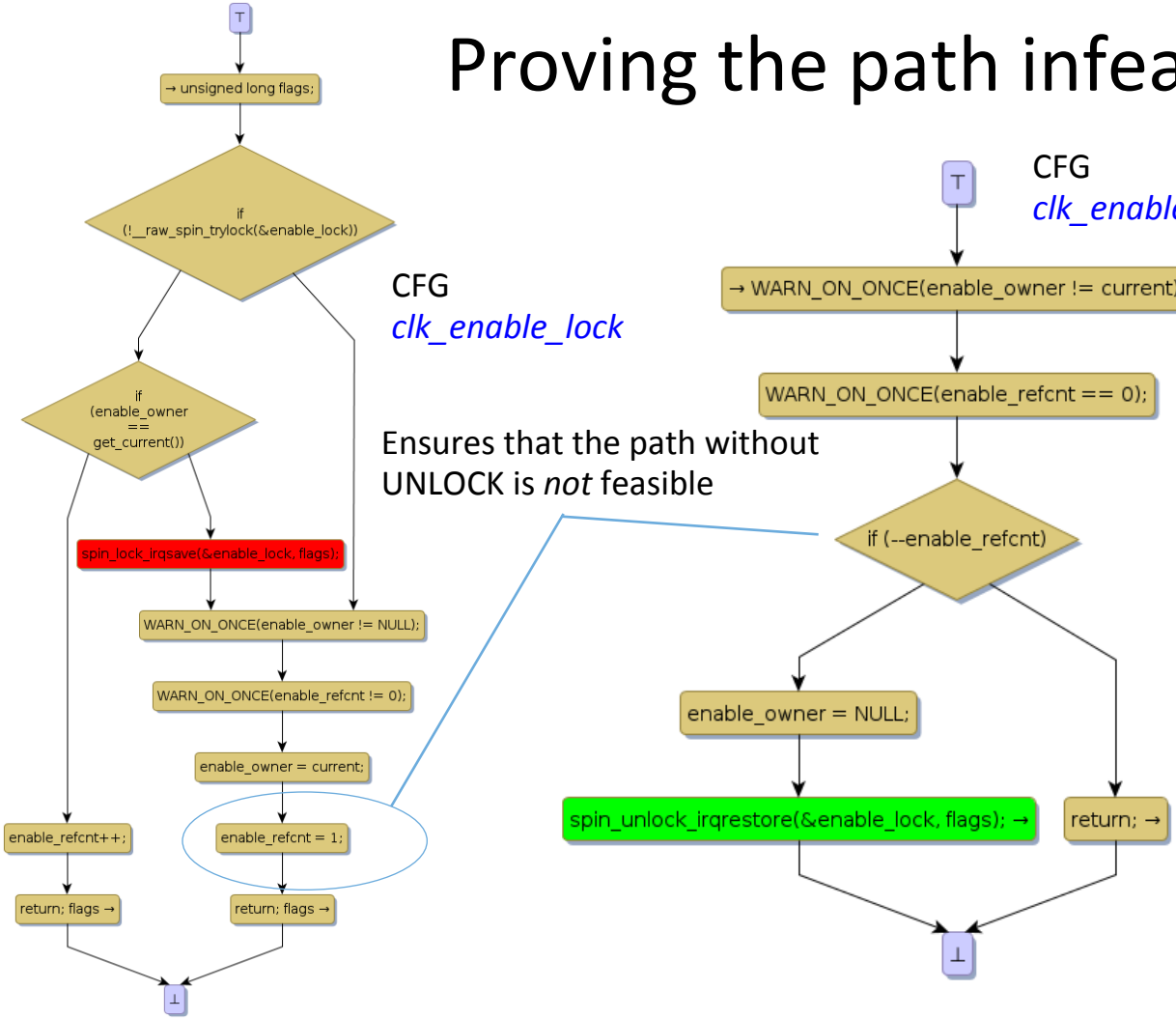


EFG *clk_enable*



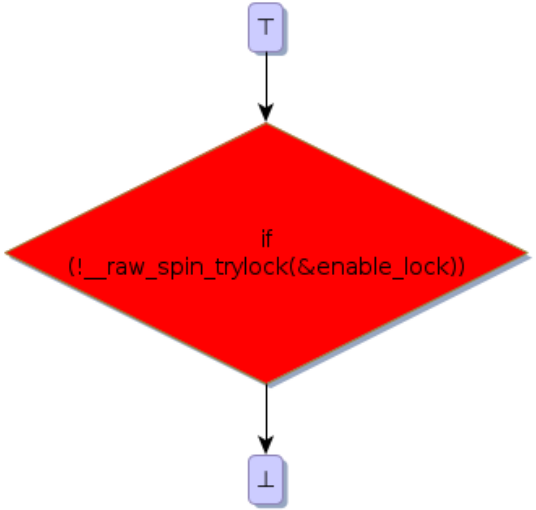
1. *clk_enable* calls *clk_enable_lock* followed by *clk_enable_unlock*.
2. However, *clk_enable_unlock* has UNLOCK on only one path.
3. We need to prove that the other path is infeasible.

Proving the path infeasibility

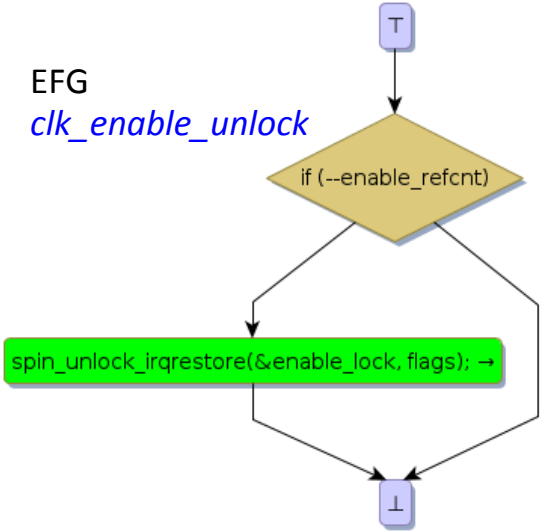


Need for the path feasibility check for the second LOCK in clk_enable_lock

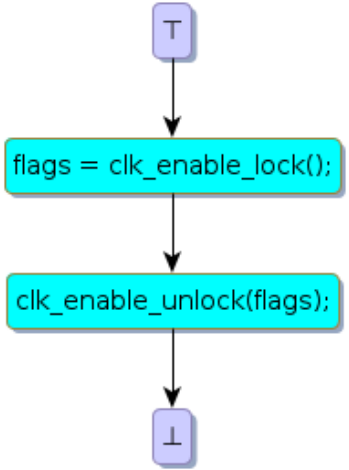
EFG *clk_enable_lock*



EFG *clk_enable_unlock*



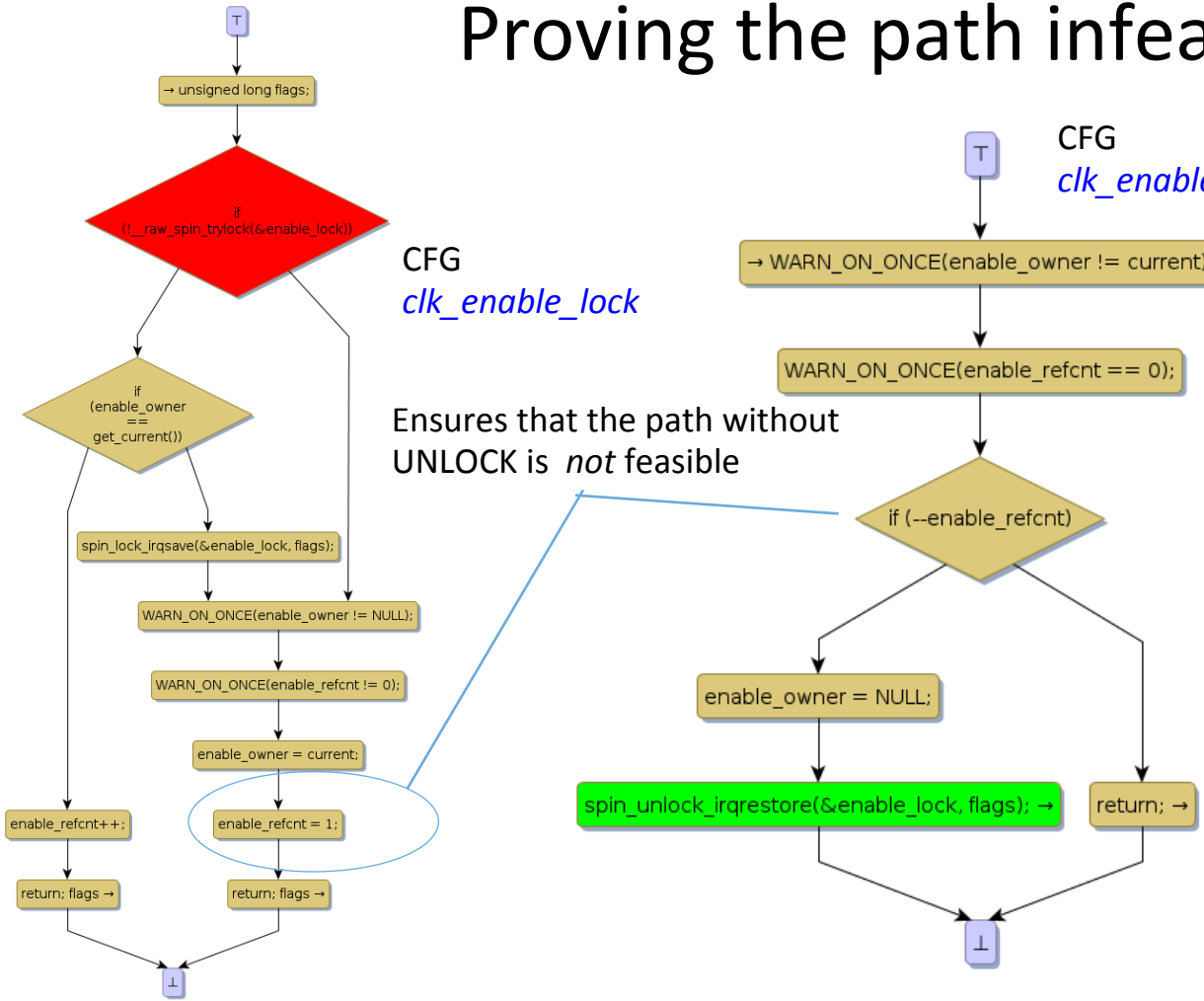
EFG *clk_enable*



1.clk_enable calls *clk_enable_lock* followed by *clk_enable_unlock*.

2. However, *clk_enable_unlock* has UNLOCK on only one path.
3. We need to prove that the other path is infeasible.

Proving the path infeasibility



CFG
clk_enable_lock

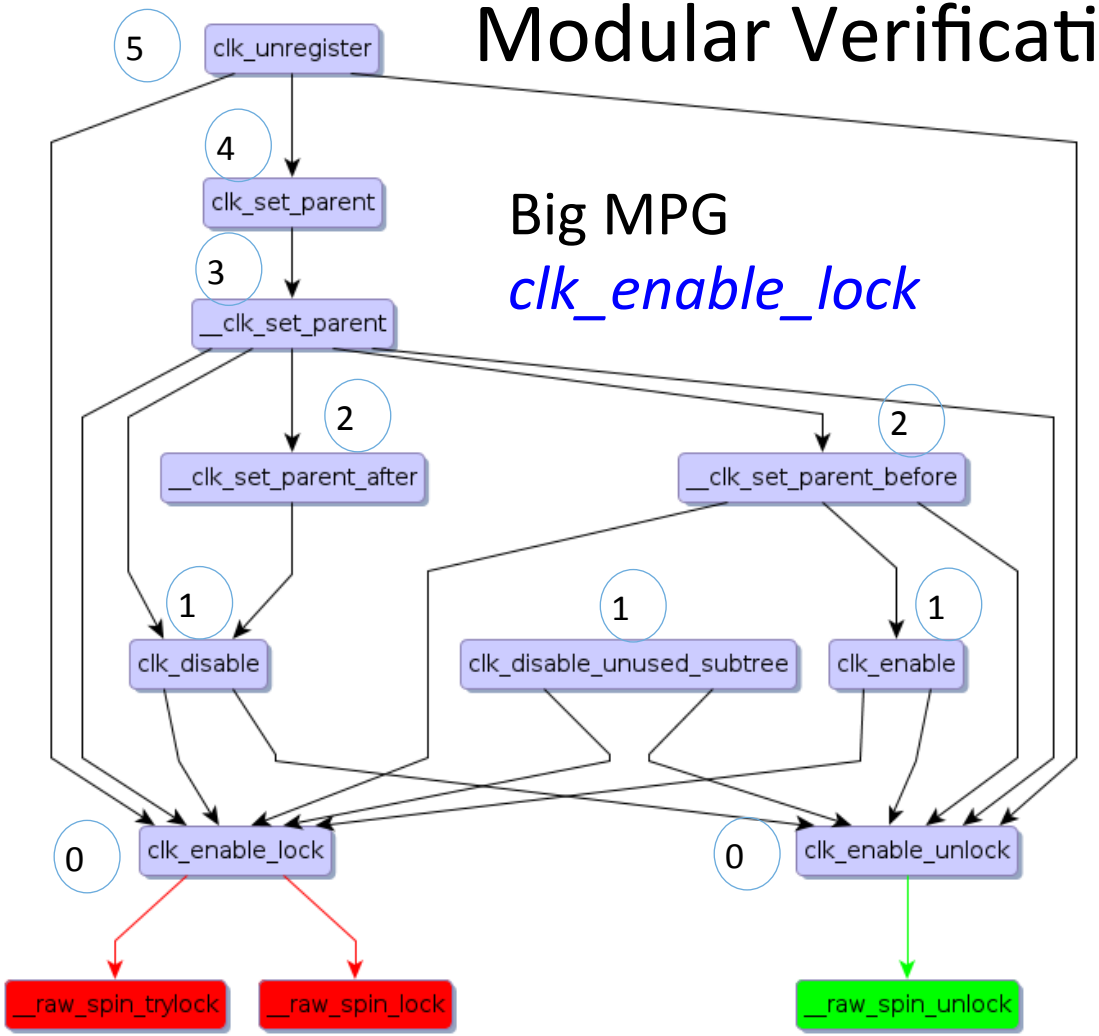
CFG
clk_enable_unlock

Ensures that the path without UNLOCK is *not* feasible

We have shown that the two LOCKS in *clk_enable_lock* are verified if *clk_enable_lock* is followed by *clk_enable_unlock*

Thus, the details of verification are encapsulated by *clk_enable_lock* and *clk_enable_unlock*. They can be used as LOCK and UNLOCK without having to check path feasibility again.

Modular Verification

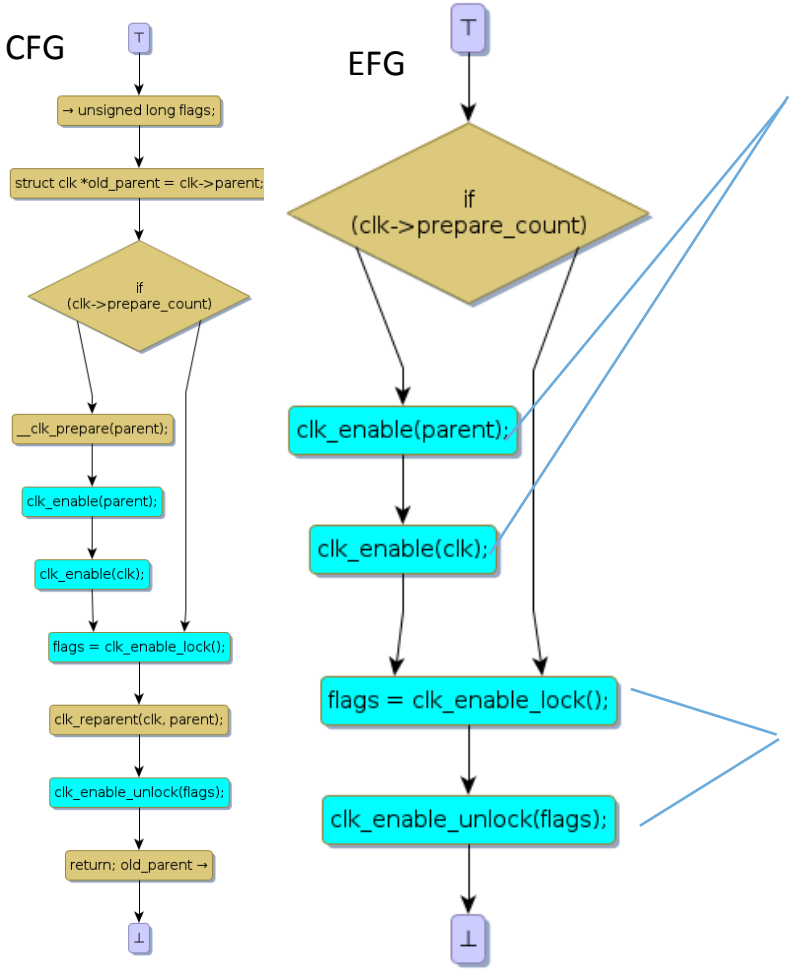


Modular Verification: *f* calls *g*. Verify *g*. Use the *g* as already verified while verifying *f*.

Modular verification based on the MPG:

1. First, verify *clk_enable*, *clk_disable*, and *clk_disable_unused_subtree are verified* - they all have *clk_enable_lock* followed by *clk_enable_unlock*
2. Next, verify *_clk_set_parent_before* and *_clk_set_parent_after*
3. Next, verify *_clk_set_parent*
4. Next, verify *clk_set_parent*
5. Next, verify *clk_unregister*

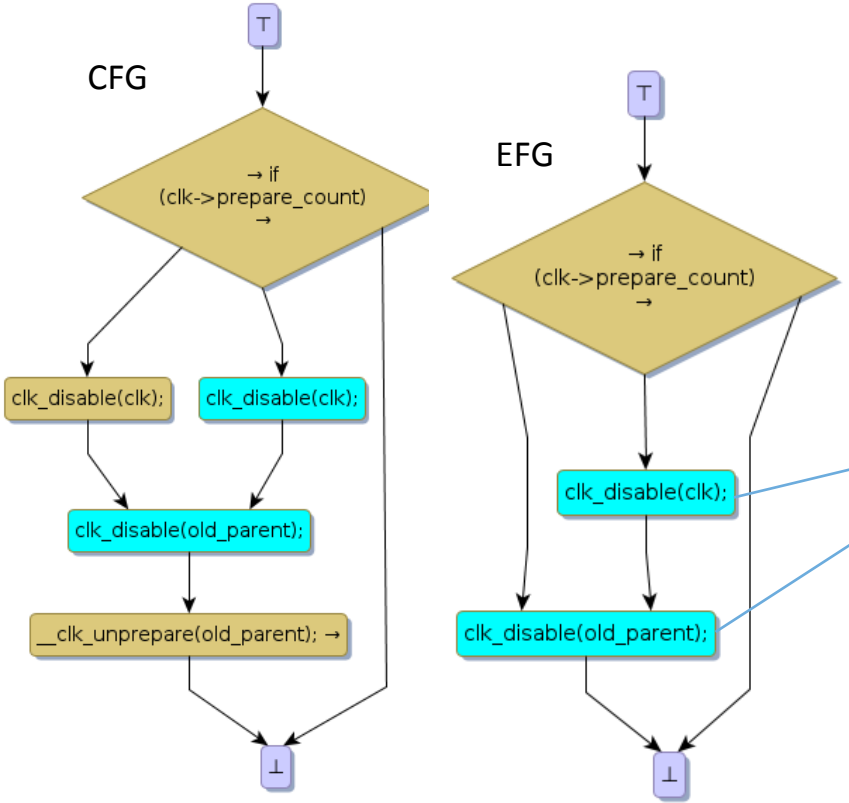
Modular Verification of *_clk_set_parent_before*



clk_enable already verified – calls *clk_enable_lock* followed by *clk_enable_unlock*

clk_enable_lock followed by *clk_enable_unlock*

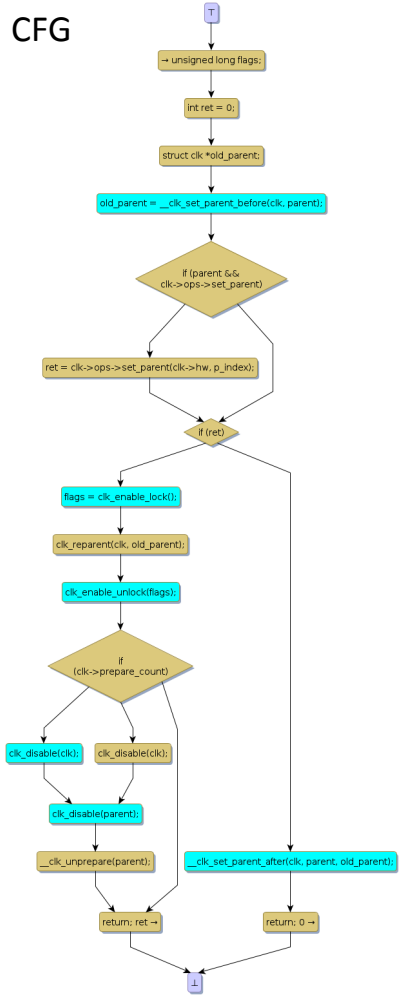
Modular Verification of *_clk_set_parent_after*



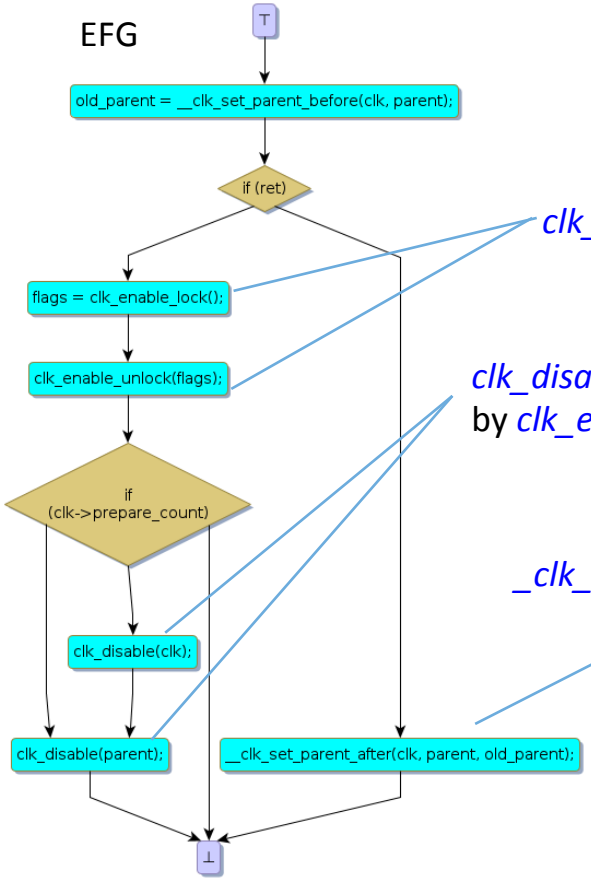
clk_disable already verified – calls *clk_enable_lock* followed by *clk_enable_unlock*

Modular Verification of *_clk_set_parent*

CFG



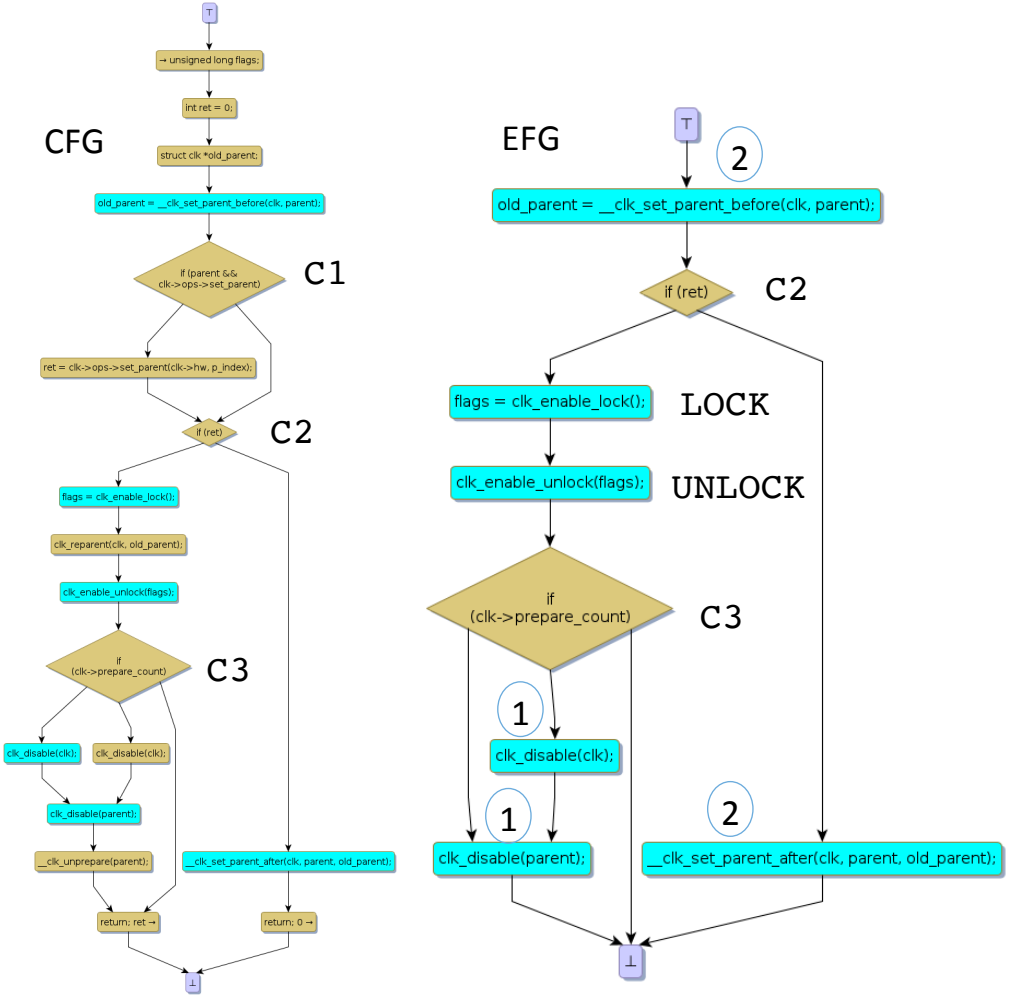
EFG



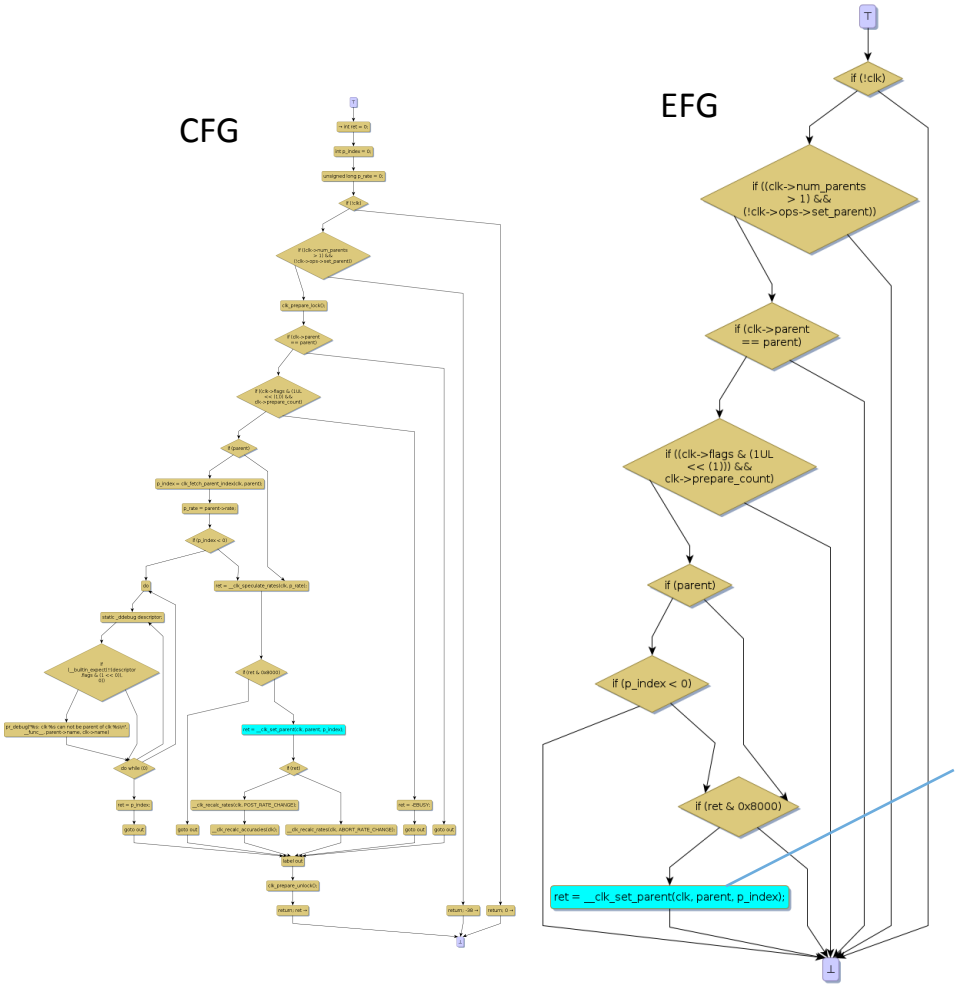
clk_enable_lock followed by *clk_enable_unlock*

clk_disable already verified – calls *clk_enable_lock* followed by *clk_enable_unlock*

_clk_set_parent_after already verified

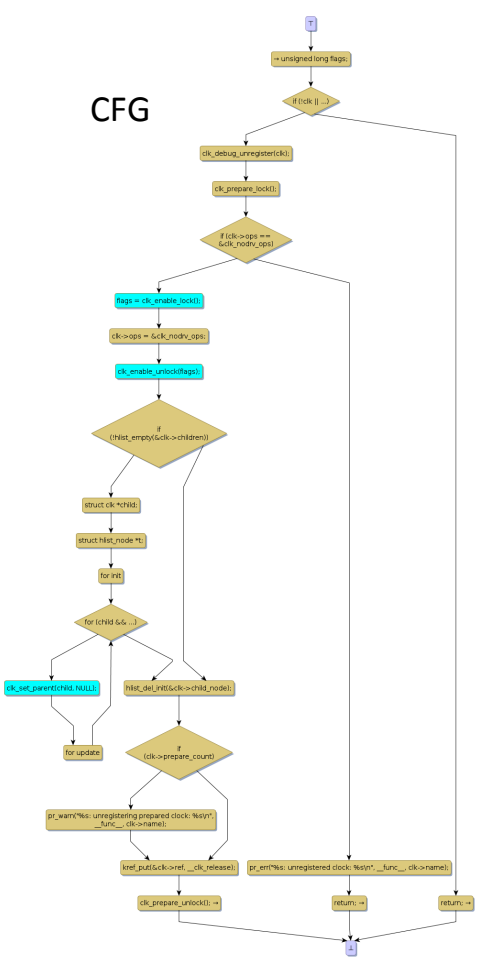

CFG
T
→ unsigned long flags;
int ret = 0;
struct clk *old_parent;
old_parent = __clk_set_parent_before(clk, parent);
if (parent &&
clk->ops->set_parent)
C1
ret = clk->ops->set_parent(clk->hw, p_index);
if (ret)
C2
flags = clk_enable_lock();
clk_reparent(clk, old_parent);
clk_enable_unlock(flags);
if
(clk->prepare_count)
C3
clk_disable(clk);
clk_disable(clk);
clk_disable(parent);
__clk_unprepare(parent);
__clk_set_parent_after(clk, parent, old_parent);
return; ret →
return; 0 →
⊥
EFG
T
2
old_parent = __clk_set_parent_before(clk, parent);
if (ret)
C2
flags = clk_enable_lock();
LOCK
clk_enable_unlock(flags);
UNLOCK
if
(clk->prepare_count)
C3
1
clk_disable(clk);
1
clk_disable(parent);
2
__clk_set_parent_after(clk, parent, old_parent);
⊥

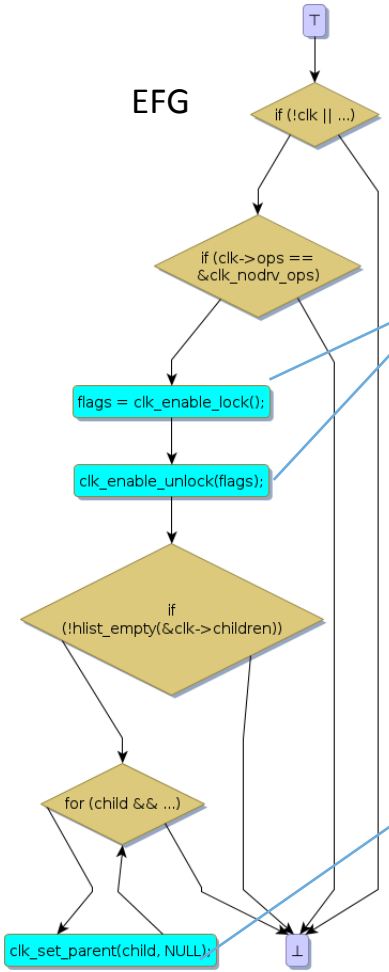
Modular Verification of *clk_set_parent*



_clk_set_parent already verified

Modular Verification of *clk_unregister*





clk_enable_lock followed by *clk_enable_unlock*

clk_set_parent already verified