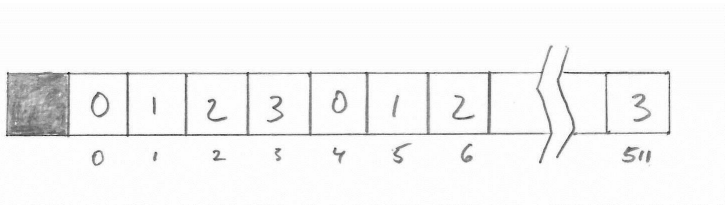
Exhaustive RRC Schedule Configurator

Mika Nyström

RRC Schedule

- Order in which ingress ports and egress TXQs are polled
- A list of up to 512 explicit entries and one implied, naming ports
- Schedule run through at constant speed (one step per cycle)



Goals

- Must satisfy bandwidth requirement of ports
- Must satisfy various constraints emanating from design details
- Goal of this work: be able to generate schedule that satisfies any satisfiable constraints
- ... at least in principle

Constraints

- Introduce `ch`: $\texttt{ch}[p, s] \leftrightarrow$ port `p` serviced in schedule slot s
- Can now write constraints in this form, e.g., port `p` not serviced more often than once every K cycles:

$$\forall s : 0 \leq s \leq 511 : \Big(\forall t : t < (s + K) \mod 513 : \texttt{ch}[\texttt{p}, s] \Rightarrow \neg \texttt{ch}[\texttt{p}, t]\Big)$$

- Represent `ch` as array of BDDs and write constraints as BDD formulas (e.g., above formula turns into a doubly nested for loop on initialization)
- The constraints of the problem *plus* assertions that you get a complete solution
- Search space exhaustively: program goes in order of increasing schedule length L, then lexicographic order among all schedules of length L

```
PROCEDURE AssertNoSmallerGap(sp  : CARDINAL;
                             gap : LONGREAL) =
  BEGIN
    FOR sl := 0 TO nslots-1 DO
      FOR j := sl+1 TO sl + CEILING(gap)-1 DO
        Assert(Implies(ch[sp,sl], Not(ch[sp,j MOD nslots])));
      END
    END
  END AssertNoSmallerGap;
```

Optimizations

- Pre-calculate that a whole branch will not succeed: for instance, that no assignment of N_p ports to N_s slots can satisfy the given port speeds

- Database of assertions maintains links from BDD literals to assertion expressions (don't need to re-evaluate entire database for each update)

- Assertions are "reduced" so that there is at most one per dependency set

- Replace boolean representation with code that only searches those schedules that satisfies given constraint (more or less easy)

Example

```
CLOCK         698.11e6
N             512
UNDERRUNMAX     2.0
MINPORTREP      4
DELTAGAPMAX     3
MINREPBITS    672.0
PORT 100.0e9
PORT 100.0e9
PORT 100.0e9
PORT 100.0e9
PORT  10.0e9
```

Experiences

- Works well for simple examples...
- Works well for some types of corner cases, very flexible
- Works less well for long schedules (slow ports)
- Still exponential and slow
- 463 Gbps schedule took 3 days to generate
- BDDs are not "industrial grade" for this problem
- Lexicographically first schedule is not interesting
- Doesn't handle PhysPort, overconstrains jitter $> 10G$

Recommendations/Conclusions

- Use when special flexibility req'd
- Use only for special cases, save schedule?
- If there is interest in having "perfect" configurator, possible program could be optimized (ditch BDDs)