MODULE *Device*

EXTENDS *Naturals*, *FiniteSets*, *Sequences*, *Messages*

Device states

CONSTANTS *Running*, *Stopped*

The following variables are used by the device to track *mastership*.

The current state of the device, either *Running* or *Stopped*

VARIABLE *state*

A mapping of stream election *IDs*

VARIABLE *election*

A mapping of stream epochs

VARIABLE *epoch*

The epoch of the last successful write to the device

VARIABLE *maxEpoch*

The following variables are used for model checking.

A history of successful writes to the switch used for model checking

VARIABLE *history*

Device related variables

$deviceVars \triangleq \langle state, election, epoch, maxEpoch, history \rangle$

Device state related variables

$stateVars \triangleq \langle state \rangle$

This section models a *P4* Runtime device. For the purposes of this spec, the device has two functions: determine a master controller node and accept writes. Mastership is determined through *MasterArbitrationUpdates* sent by the controller nodes. The 'election_id's provided by controller nodes are stored in 'elections', and the master is computed as the node with the highest 'election_id' at any given time. The device will only allow writes from the current master node.

Returns the set of election *IDs* in the given elections

$ElectionIds(e) \triangleq \{e[x] : x \in \text{DOMAIN } e\}$

Returns the maximum value from a set or undefined if the set is empty

$Max(s) \triangleq \text{CHOOSE } x \in s : \forall\, y \in s : x \geq y$

Returns the highest election *ID* for the given elections

$MaxElectionId(e) \triangleq Max(ElectionIds(e))$

Returns the master for the given elections

$MasterId(e) \triangleq$
IF $Cardinality(\{i \in ElectionIds(e) : i > 0\}) > 0$ THEN
CHOOSE $n \in$ DOMAIN $e : e[n] = MaxElectionId(e)$
ELSE
Nil

Shuts down the device

When the device is shutdown, all the volatile device and stream variables are set back to their initial state. The 'maxEpoch' accepted by the device is persisted through the restart.

$Shutdown \triangleq$
$\land\ state = Running$
$\land\ state' = Stopped$
$\land\ responseStream' = [n \in$ DOMAIN $responseStream \mapsto [id \mapsto responseStream[n].id,\ state \mapsto Closed]]$
$\land\ requests' = [n \in$ DOMAIN $requests \mapsto \langle\rangle]$
$\land\ responses' = [n \in$ DOMAIN $responses \mapsto \langle\rangle]$
$\land\ election' = [n \in$ DOMAIN $election \mapsto 0]$
$\land\ epoch' = [n \in$ DOMAIN $epoch \mapsto 0]$
$\land$ UNCHANGED $\langle maxEpoch,\ requestStream,\ history\rangle$

Starts the device

$Startup \triangleq$
$\land\ state = Stopped$
$\land\ state' = Running$
$\land$ UNCHANGED $\langle messageVars,\ election,\ epoch,\ maxEpoch,\ history,\ streamVars\rangle$

Opens a new stream between node 'n' and the device

When a stream is opened, the 'streams' state for node 'n' is set to *Open*. Stream creation is modelled as a single step to reduce the state space.

$ConnectStream(n) \triangleq$
$\land\ state = Running$
$\land\ requestStream[n].state = Open$
$\land\ responseStream[n].id < requestStream[n].id$
$\land\ responseStream[n].state = Closed$
$\land\ responseStream' = [responseStream$ EXCEPT $![n].state = Open]$
$\land$ UNCHANGED $\langle deviceVars,\ messageVars,\ requestStream\rangle$

Closes an open stream between node 'n' and the device

When a stream is closed, the 'streams' state for node 'n' is set to *Closed*, any 'election_id' provided by node 'n' is forgotten, and the 'requests' and 'responses' queues for the node are cleared. Additionally, if the stream belonged to the master node, a new master is elected and a *MasterArbitrationUpdate* is sent on the streams that remain in the *Open* state. The *MasterArbitrationUpdate* will be sent to the new master with a 'status' of *Ok* and to all slaves with a 'status' of *AlreadyExists*.

$DisconnectStream(n) \triangleq$
$\land\ state = Running$
$\land\ responseStream[n].state = Open$

```
    ∧ election′ = [election EXCEPT ![n] = 0]
    ∧ epoch′ = [epoch EXCEPT ![n] = 0]
    ∧ responseStream′ = [responseStream EXCEPT ![n].state = Closed]
    ∧ requests′ = [requests EXCEPT ![n] = ⟨⟩]
    ∧ LET oldMaster ≜ MasterId(election)
          newMaster ≜ MasterId(election′)
      IN
          ∨ ∧ oldMaster ≠ newMaster
            ∧ responses′ = [i ∈ DOMAIN responseStream′ ↦
                              IF responseStream′[i].state = Open THEN
                                  IF i = newMaster THEN
                                      Append(responses[i], [
                                          type        ↦ MasterArbitrationUpdate,
                                          status      ↦ Ok,
                                          election_id ↦ MaxElectionId(election′)])
                                   ELSE
                                      Append(responses[i], [
                                          type        ↦ MasterArbitrationUpdate,
                                          status      ↦ AlreadyExists,
                                          election_id ↦ MaxElectionId(election′)])
                                ELSE
                                  ⟨⟩]
          ∨ ∧ oldMaster = newMaster
            ∧ responses′ = [responses EXCEPT ![n] = ⟨⟩]
    ∧ UNCHANGED ⟨stateVars, maxEpoch, requestStream, history⟩
```

The device receives and responds to a *MasterArbitrationUpdate* from node 'n'

If the 'election_id' is already present in the 'elections' and does not already belong to node 'n', the stream is *Closed* and 'requests' and 'responses' are cleared for the node. If the 'election_id' is not known to the device, it's added to the 'elections' state. If the change results in a new master being elected by the device, a *MasterArbitrationUpdate* is sent on all *Open* streams. If the change does not result in a new master being elected by the device, node 'n' is returned a *MasterArbitrationUpdate*. The device master will always receive a *MasterArbitrationUpdate* response with 'status' of *Ok*, and slaves will always receive a 'status' of *AlreadyExists*.

```
HandleMasterArbitrationUpdate(n) ≜
    ∧ state = Running
    ∧ responseStream[n].state = Open
    ∧ HasRequest(n, MasterArbitrationUpdate)
    ∧ LET r ≜ NextRequest(n)
      IN
          ∨ ∧ r.election_id ∈ ElectionIds(election)
            ∧ election[n] ≠ r.election_id
            ∧ responseStream′ = [responseStream EXCEPT ![n].state = Closed]
            ∧ requests′ = [requests EXCEPT ![n] = ⟨⟩]
            ∧ responses′ = [responses EXCEPT ![n] = ⟨⟩]
```

```
    ∧ UNCHANGED ⟨deviceVars⟩
  ∨ ∧ r.election_id ∉ ElectionIds(election)
    ∧ election′ = [election EXCEPT ![n] = r.election_id]
    ∧ epoch′ = [epoch EXCEPT ![n] = r.epoch]
    ∧ LET oldMaster ≜ MasterId(election)
          newMaster ≜ MasterId(election′)
      IN
          ∨ ∧ oldMaster ≠ newMaster
            ∧ responses′ = [i ∈ DOMAIN responseStream ↦
                              IF responseStream[i].state = Open THEN
                                  IF i = newMaster THEN
                                      Append(responses[i], [
                                          type        ↦ MasterArbitrationUpdate,
                                          status      ↦ Ok,
                                          election_id ↦ MaxElectionId(election′)])
                                  ELSE
                                      Append(responses[i], [
                                          type        ↦ MasterArbitrationUpdate,
                                          status      ↦ AlreadyExists,
                                          election_id ↦ MaxElectionId(election′)])
                              ELSE
                                  responses[i]]
          ∨ ∧ oldMaster = newMaster
            ∧ ∨ ∧ n = newMaster
                ∧ SendResponse(n, [
                      type        ↦ MasterArbitrationUpdate,
                      status      ↦ Ok,
                      election_id ↦ MaxElectionId(election′)])
              ∨ ∧ n ≠ newMaster
                ∧ SendResponse(n, [
                      type        ↦ MasterArbitrationUpdate,
                      status      ↦ AlreadyExists,
                      election_id ↦ MaxElectionId(election′)])
            ∧ UNCHANGED ⟨responseStream⟩
  ∧ DiscardRequest(n)
  ∧ UNCHANGED ⟨stateVars, maxEpoch, requestStream, history⟩
```

The device receives a *WriteRequest* from node 'n'

The *WriteRequest* is accepted if:
* The 'election_id' for node 'n' matches the 'election_id' for its stream
* Node 'n' is the current master for the device
* If node 'n' provided an 'epoch' and the 'epoch' is greater than or equal to the highest epoch received by the device

When the *WriteRequest* is accepted, the 'maxEpoch' is updated and the term of the node that sent the request is recorded for model checking. If the *WriteRequest* is rejeceted, a *PermissionDenied* response is returned.

```
HandleWrite(n) ≜
    ∧ state = Running
    ∧ responseStream[n].state = Open
    ∧ HasRequest(n, WriteRequest)
    ∧ LET r ≜ NextRequest(n)
      IN
          ∨ ∧ election[n] = r.election_id
            ∧ MasterId(election) = n
            ∧ epoch[n] > 0 ⇒ epoch[n] ≥ maxEpoch
            ∧ maxEpoch' = epoch[n]
            ∧ history' = Append(history, [node ↦ n, term ↦ r.term])
            ∧ SendResponse(n, [
                  type   ↦ WriteResponse,
                  status ↦ Ok])
          ∨ ∧ ∨ election[n] ≠ r.election_id
              ∨ MasterId(election) ≠ n
              ∨ ∧ epoch[n] > 0
                ∧ epoch[n] < maxEpoch
            ∧ SendResponse(n, [
                  type   ↦ WriteResponse,
                  status ↦ PermissionDenied])
            ∧ UNCHANGED ⟨maxEpoch, history⟩
    ∧ DiscardRequest(n)
    ∧ UNCHANGED ⟨stateVars, election, epoch, streamVars⟩
```

* Modification History
* Last modified *Thu Feb* 21 16:59:22 *PST* 2019 by *jordanhalterman*
* Created *Wed Feb* 20 23:49:17 *PST* 2019 by *jordanhalterman*