──────────────────── 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*

The last successful write token
VARIABLE *writeToken*

├──┤

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
$deviceVars \triangleq \langle state, \ election, \ writeToken, \ history \rangle$

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
$MasterId(e) \triangleq$

1

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

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

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

$Startup \triangleq$
   $\wedge\ state = Stopped$
   $\wedge\ state' = Running$
   $\wedge$ UNCHANGED $\langle messageVars,\ election,\ writeToken,\ history,\ streamVars\rangle$

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

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

When a stream is disconnected, 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 $AlreadyExists$.

$DisconnectStream(n) \triangleq$
   $\wedge\ state = Running$
   $\wedge\ responseStream[n].state = Open$
   $\wedge\ election' = [election$ EXCEPT $![n] = 0]$
   $\wedge\ responseStream' = [responseStream$ EXCEPT $![n].state = Closed]$
   $\wedge\ requests' = [requests$ EXCEPT $![n] = \langle\rangle]$

$\wedge$ LET $oldMaster \triangleq MasterId(election)$
$\qquad newMaster \triangleq MasterId(election')$

$\quad$ IN

$\qquad \vee \wedge oldMaster \neq newMaster$
$\qquad \quad \wedge responses' = [i \in \text{DOMAIN } responseStream' \mapsto$
$\qquad \qquad \qquad \qquad \qquad \text{IF } responseStream'[i].state = Open \text{ THEN}$
$\qquad \qquad \qquad \qquad \qquad \quad \text{IF } i = newMaster \text{ THEN}$
$\qquad \qquad \qquad \qquad \qquad \qquad Append(responses[i], [$
$\qquad \qquad \qquad \qquad \qquad \qquad \quad type \qquad \mapsto MasterArbitrationUpdate,$
$\qquad \qquad \qquad \qquad \qquad \qquad \quad status \qquad \mapsto Ok,$
$\qquad \qquad \qquad \qquad \qquad \qquad \quad election\_id \mapsto MaxElectionId(election')])$
$\qquad \qquad \qquad \qquad \qquad \quad \text{ELSE}$
$\qquad \qquad \qquad \qquad \qquad \qquad Append(responses[i], [$
$\qquad \qquad \qquad \qquad \qquad \qquad \quad type \qquad \mapsto MasterArbitrationUpdate,$
$\qquad \qquad \qquad \qquad \qquad \qquad \quad status \qquad \mapsto AlreadyExists,$
$\qquad \qquad \qquad \qquad \qquad \qquad \quad election\_id \mapsto MaxElectionId(election')])$
$\qquad \qquad \qquad \qquad \qquad \text{ELSE}$
$\qquad \qquad \qquad \qquad \qquad \quad \langle\rangle]$
$\qquad \vee \wedge oldMaster = newMaster$
$\qquad \quad \wedge responses' = [responses \text{ EXCEPT } ![n] = \langle\rangle]$
$\wedge \text{UNCHANGED } \langle stateVars, writeToken, requestStream, history\rangle$

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*.

$HandleMasterArbitrationUpdate(n) \triangleq$
$\quad \wedge state = Running$
$\quad \wedge responseStream[n].state = Open$
$\quad \wedge HasRequest(n, MasterArbitrationUpdate)$
$\quad \wedge \text{LET } r \triangleq NextRequest(n)$

$\qquad \text{IN}$

$\qquad \quad \vee \wedge r.election\_id \in ElectionIds(election)$
$\qquad \qquad \wedge election[n] \neq r.election\_id$
$\qquad \qquad \wedge responseStream' = [responseStream \text{ EXCEPT } ![n].state = Closed]$
$\qquad \qquad \wedge requests' = [requests \text{ EXCEPT } ![n] = \langle\rangle]$
$\qquad \qquad \wedge responses' = [responses \text{ EXCEPT } ![n] = \langle\rangle]$
$\qquad \qquad \wedge \text{UNCHANGED } \langle deviceVars\rangle$
$\qquad \quad \vee \wedge r.election\_id \notin ElectionIds(election)$
$\qquad \qquad \wedge election' = [election \text{ EXCEPT } ![n] = r.election\_id]$
$\qquad \qquad \wedge \text{LET } oldMaster \triangleq MasterId(election)$

$$newMaster \triangleq MasterId(election')$$

IN

$\lor \ \land \ oldMaster \neq newMaster$

　　$\land \ responses' = [i \in \text{DOMAIN } responseStream \mapsto$

　　　　　　　IF $responseStream[i].state = Open$ THEN

　　　　　　　　　IF $i = newMaster$ THEN

　　　　　　　　　　　$Append(responses[i], [$

　　　　　　　　　　　　　$type \qquad \mapsto MasterArbitrationUpdate,$

　　　　　　　　　　　　　$status \quad \mapsto Ok,$

　　　　　　　　　　　　　$election\_id \mapsto MaxElectionId(election')])$

　　　　　　　　　ELSE

　　　　　　　　　　　$Append(responses[i], [$

　　　　　　　　　　　　　$type \qquad \mapsto MasterArbitrationUpdate,$

　　　　　　　　　　　　　$status \quad \mapsto AlreadyExists,$

　　　　　　　　　　　　　$election\_id \mapsto MaxElectionId(election')])$

　　　　　　　ELSE

　　　　　　　　　$responses[i]]$

$\lor \ \land \ oldMaster = newMaster$

　　$\land \ \lor \ \land \ n = newMaster$

　　　　　$\land \ SendResponse(n, [$

　　　　　　　$type \qquad \mapsto MasterArbitrationUpdate,$

　　　　　　　$status \quad \mapsto Ok,$

　　　　　　　$election\_id \mapsto MaxElectionId(election')])$

　　　$\lor \ \land \ n \neq newMaster$

　　　　　$\land \ SendResponse(n, [$

　　　　　　　$type \qquad \mapsto MasterArbitrationUpdate,$

　　　　　　　$status \quad \mapsto AlreadyExists,$

　　　　　　　$election\_id \mapsto MaxElectionId(election')])$

$\land \ \text{UNCHANGED } \langle responseStream \rangle$

$\land \ DiscardRequest(n)$

$\land \ \text{UNCHANGED } \langle stateVars, writeToken, requestStream, history \rangle$

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 a 'token' is provided in the *WroteRequest* and the 'token' is greater than or equal to the last 'writeToken' accepted by the device

When the *WriteRequest* is accepted, the 'writeToken' 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.

$HandleWrite(n) \triangleq$

　$\land \ state = Running$

　$\land \ responseStream[n].state = Open$

　$\land \ HasRequest(n, WriteRequest)$

　$\land \ \text{LET } r \triangleq NextRequest(n)$

4

IN
  ∨ ∧ $election[n] = r.election\_id$
    ∧ $MasterId(election) = n$
    ∧ $r.token > 0 \Rightarrow r.token \geq writeToken$
    ∧ $writeToken' = r.token$
    ∧ $history' = Append(history, [node \mapsto n, term \mapsto r.term])$
    ∧ $SendResponse(n, [$
      $type \quad \mapsto WriteResponse,$
      $status \mapsto Ok])$
  ∨ ∧ ∨ $election[n] \neq r.election\_id$
     ∨ $MasterId(election) \neq n$
     ∨ $r.token = 0$
     ∨ $r.token < writeToken$
    ∧ $SendResponse(n, [$
      $type \quad \mapsto WriteResponse,$
      $status \mapsto PermissionDenied])$
    ∧ UNCHANGED $\langle writeToken, history \rangle$
 ∧ $DiscardRequest(n)$
 ∧ UNCHANGED $\langle stateVars, election, streamVars \rangle$

\ * Modification History
\ * Last modified *Mon Feb* 25 16:27:03 *PST* 2019 by *jordanhalterman*
\ * Created *Wed Feb* 20 23:49:17 *PST* 2019 by *jordanhalterman*