———————————————— 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*

The election *ID* of the highest master
VARIABLE *electionId*

The current master node
VARIABLE *currentMaster*

⊢——

The following variables are used for model checking.

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

⊢——

Election related variables
$electionVars \triangleq \langle electionId,\ currentMaster \rangle$

Device related variables
$deviceVars \triangleq \langle state,\ history,\ electionVars \rangle$

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.

Shuts down the device

When the device is shutdown, all the volatile device and stream variables are set back to their initial state. The 'writeTerm' accepted by the device is persisted through the restart.
$Shutdown \triangleq$
$\quad \wedge state = Running$
$\quad \wedge state' = Stopped$
$\quad \wedge responseStream' = [n \in \text{DOMAIN}\ responseStream \mapsto [id \mapsto responseStream[n].id,\ state \mapsto Closed]]$

1

$\wedge\ requests' = [n \in \text{DOMAIN}\ requests \mapsto \langle\rangle]$
$\wedge\ responses' = [n \in \text{DOMAIN}\ responses \mapsto \langle\rangle]$
$\wedge\ \text{UNCHANGED}\ \langle requestStream,\ electionVars,\ history\rangle$

Starts the device

$Startup\ \triangleq$
   $\wedge\ state = Stopped$
   $\wedge\ state' = Running$
   $\wedge\ \text{UNCHANGED}\ \langle messageVars,\ electionVars,\ history,\ streamVars\rangle$

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

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.

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

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

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*.

$DisconnectStream(n)\ \triangleq$
   $\wedge\ state = Running$
   $\wedge\ responseStream[n].state = Open$
   $\wedge\ responseStream' = [responseStream\ \text{EXCEPT}\ ![n].state = Closed]$
   $\wedge\ requests' = [requests\ \text{EXCEPT}\ ![n] = \langle\rangle]$
   $\wedge\ responses' = [responses\ \text{EXCEPT}\ ![n] = \langle\rangle]$
   $\wedge\ \text{UNCHANGED}\ \langle stateVars,\ requestStream,\ electionVars,\ history\rangle$

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*.

$HandleMasterArbitrationUpdate(n)\ \triangleq$
   $\wedge\ state = Running$
   $\wedge\ responseStream[n].state = Open$

2

$\wedge\ HasRequest(n,\ MasterArbitrationUpdate)$
$\wedge\ \text{LET}\ r\ \triangleq\ NextRequest(n)$
$\quad\text{IN}$
$\qquad\vee\ \wedge\ r.election\_id\quad =\ electionId$
$\qquad\quad\wedge\ currentMaster \neq n$
$\qquad\quad\wedge\ responseStream' = [responseStream\ \text{EXCEPT}\ ![n].state = Closed]$
$\qquad\quad\wedge\ requests' = [requests\ \text{EXCEPT}\ ![n] = \langle\rangle]$
$\qquad\quad\wedge\ responses' = [responses\ \text{EXCEPT}\ ![n] = \langle\rangle]$
$\qquad\quad\wedge\ \text{UNCHANGED}\ \langle deviceVars\rangle$
$\qquad\vee\ \wedge\ r.election\_id > electionId$
$\qquad\quad\wedge\ electionId' = r.election\_id$
$\qquad\quad\wedge\ currentMaster' = n$
$\qquad\quad\wedge\ responses' = [i \in \text{DOMAIN}\ responseStream \mapsto$
$\qquad\qquad\qquad\qquad\quad\text{IF}\ responseStream[i].state = Open\ \text{THEN}$
$\qquad\qquad\qquad\qquad\quad\ Append(responses[i],\ [$
$\qquad\qquad\qquad\qquad\qquad\ type\qquad\ \mapsto MasterArbitrationUpdate,$
$\qquad\qquad\qquad\qquad\qquad\ status\quad\ \mapsto Ok,$
$\qquad\qquad\qquad\qquad\qquad\ election\_id \mapsto r.election\_id])$
$\qquad\qquad\qquad\qquad\quad\text{ELSE}$
$\qquad\qquad\qquad\qquad\qquad\ responses[i]]$
$\qquad\quad\wedge\ \text{UNCHANGED}\ \langle responseStream\rangle$
$\qquad\vee\ \wedge\ r.election\_id < electionId$
$\qquad\quad\wedge\ SendResponse(n,\ [$
$\qquad\qquad\ type\qquad\ \mapsto MasterArbitrationUpdate,$
$\qquad\qquad\ status\quad\ \mapsto AlreadyExists,$
$\qquad\qquad\ election\_id \mapsto electionId])$
$\qquad\quad\wedge\ \text{UNCHANGED}\ \langle deviceVars,\ responseStream\rangle$
$\wedge\ DiscardRequest(n)$
$\wedge\ \text{UNCHANGED}\ \langle stateVars,\ requestStream,\ history\rangle$

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.

$HandleWrite(n)\ \triangleq$
$\quad\wedge\ state = Running$
$\quad\wedge\ responseStream[n].state = Open$
$\quad\wedge\ HasRequest(n,\ WriteRequest)$
$\quad\wedge\ \text{LET}\ r\ \triangleq\ NextRequest(n)$
$\qquad\text{IN}$
$\qquad\quad\vee\ \wedge\ r.election\_id\quad =\ electionId$

$$\land \mathit{currentMaster} = n$$
$$\land \mathit{history}' = \mathit{Append}(\mathit{history}, [\mathit{node} \mapsto n,\ \mathit{term} \mapsto r.\mathit{term}])$$
$$\land \mathit{SendResponse}(n, [$$
$$\quad \mathit{type} \quad \mapsto \mathit{WriteResponse},$$
$$\quad \mathit{status} \mapsto \mathit{Ok}])$$
$$\lor\ \land\ \lor\ r.\mathit{election\_id} \quad \neq \mathit{electionId}$$
$$\quad \lor\ \mathit{currentMaster} \neq n$$
$$\land \mathit{SendResponse}(n, [$$
$$\quad \mathit{type} \quad \mapsto \mathit{WriteResponse},$$
$$\quad \mathit{status} \mapsto \mathit{PermissionDenied}])$$
$$\land \textsc{unchanged}\ \langle \mathit{history} \rangle$$
$$\land \mathit{DiscardRequest}(n)$$
$$\land \textsc{unchanged}\ \langle \mathit{stateVars},\ \mathit{electionVars},\ \mathit{streamVars} \rangle$$