
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 term

VARIABLE *writeTerm*

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

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

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 'writeTerm' accepted by the device is persisted through the restart.

```

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

```

Starts the device

```

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

```

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

```

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

```

$$\begin{aligned}
& \wedge \text{LET } oldMaster \triangleq MasterId(election) \\
& \quad newMaster \triangleq MasterId(election') \\
& \text{IN} \\
& \quad \vee \wedge oldMaster \neq newMaster \\
& \quad \quad \wedge responses' = [i \in \text{DOMAIN } responseStream' \mapsto \\
& \quad \quad \quad \text{IF } responseStream'[i].state = Open \text{ THEN} \\
& \quad \quad \quad \quad \text{IF } i = newMaster \text{ THEN} \\
& \quad \quad \quad \quad \quad Append(responses[i], [\\
& \quad \quad \quad \quad \quad \quad type \mapsto MasterArbitrationUpdate, \\
& \quad \quad \quad \quad \quad \quad status \mapsto Ok, \\
& \quad \quad \quad \quad \quad \quad election_id \mapsto MaxElectionId(election')]) \\
& \quad \quad \quad \quad \text{ELSE} \\
& \quad \quad \quad \quad \quad Append(responses[i], [\\
& \quad \quad \quad \quad \quad \quad type \mapsto MasterArbitrationUpdate, \\
& \quad \quad \quad \quad \quad \quad status \mapsto AlreadyExists, \\
& \quad \quad \quad \quad \quad \quad election_id \mapsto MaxElectionId(election')]) \\
& \quad \quad \quad \quad \text{ELSE} \\
& \quad \quad \quad \quad \quad \langle \rangle] \\
& \quad \quad \quad \quad \vee \wedge oldMaster = newMaster \\
& \quad \quad \quad \quad \quad \wedge responses' = [responses \text{ EXCEPT } ![n] = \langle \rangle] \\
& \quad \wedge \text{UNCHANGED } \langle stateVars, writeTerm, requestStream, history \rangle
\end{aligned}$$

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*.

$$\begin{aligned}
& HandleMasterArbitrationUpdate(n) \triangleq \\
& \quad \wedge state = Running \\
& \quad \wedge responseStream[n].state = Open \\
& \quad \wedge HasRequest(n, MasterArbitrationUpdate) \\
& \quad \wedge \text{LET } r \triangleq NextRequest(n) \\
& \quad \text{IN} \\
& \quad \quad \vee \wedge r.election_id \in ElectionIds(election) \\
& \quad \quad \quad \wedge election[n] \neq r.election_id \\
& \quad \quad \quad \wedge responseStream' = [responseStream \text{ EXCEPT } ![n].state = Closed] \\
& \quad \quad \quad \wedge requests' = [requests \text{ EXCEPT } ![n] = \langle \rangle] \\
& \quad \quad \quad \wedge responses' = [responses \text{ EXCEPT } ![n] = \langle \rangle] \\
& \quad \quad \quad \wedge \text{UNCHANGED } \langle deviceVars \rangle \\
& \quad \quad \vee \wedge r.election_id \notin ElectionIds(election) \\
& \quad \quad \quad \wedge election' = [election \text{ EXCEPT } ![n] = r.election_id] \\
& \quad \quad \quad \wedge \text{LET } oldMaster \triangleq MasterId(election)
\end{aligned}$$

$$\begin{aligned}
& newMaster \triangleq MasterId(election') \\
& \text{IN} \\
& \vee \wedge oldMaster \neq newMaster \\
& \quad \wedge responses' = [i \in \text{DOMAIN } responseStream \mapsto \\
& \quad \quad \text{IF } responseStream[i].state = Open \text{ THEN} \\
& \quad \quad \quad \text{IF } i = newMaster \text{ THEN} \\
& \quad \quad \quad \quad Append(responses[i], [\\
& \quad \quad \quad \quad \quad type \mapsto MasterArbitrationUpdate, \\
& \quad \quad \quad \quad \quad status \mapsto Ok, \\
& \quad \quad \quad \quad \quad election_id \mapsto MaxElectionId(election')]) \\
& \quad \quad \quad \text{ELSE} \\
& \quad \quad \quad \quad Append(responses[i], [\\
& \quad \quad \quad \quad \quad type \mapsto MasterArbitrationUpdate, \\
& \quad \quad \quad \quad \quad status \mapsto AlreadyExists, \\
& \quad \quad \quad \quad \quad election_id \mapsto MaxElectionId(election')]) \\
& \quad \quad \quad \text{ELSE} \\
& \quad \quad \quad \quad responses[i]] \\
& \quad \vee \wedge oldMaster = newMaster \\
& \quad \quad \wedge \vee \wedge n = newMaster \\
& \quad \quad \quad \wedge SendResponse(n, [\\
& \quad \quad \quad \quad type \mapsto MasterArbitrationUpdate, \\
& \quad \quad \quad \quad status \mapsto Ok, \\
& \quad \quad \quad \quad election_id \mapsto MaxElectionId(election')]) \\
& \quad \quad \vee \wedge n \neq newMaster \\
& \quad \quad \quad \wedge SendResponse(n, [\\
& \quad \quad \quad \quad type \mapsto MasterArbitrationUpdate, \\
& \quad \quad \quad \quad status \mapsto AlreadyExists, \\
& \quad \quad \quad \quad election_id \mapsto MaxElectionId(election')]) \\
& \quad \wedge \text{UNCHANGED } \langle responseStream \rangle \\
& \quad \wedge DiscardRequest(n) \\
& \quad \wedge \text{UNCHANGED } \langle stateVars, writeTerm, requestStream, history \rangle
\end{aligned}$$

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 *WriteRequest* 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 rejected, a *PermissionDenied* response is returned.

HandleWrite(*n*) \triangleq

$$\begin{aligned}
& \wedge state = Running \\
& \wedge responseStream[n].state = Open \\
& \wedge HasRequest(n, WriteRequest) \\
& \wedge \text{LET } r \triangleq NextRequest(n)
\end{aligned}$$

IN

$$\begin{aligned}
& \vee \wedge election[n] = r.election_id \\
& \wedge MasterId(election) = n \\
& \wedge r.token > 0 \Rightarrow r.token \geq writeTerm \\
& \wedge writeTerm' = r.token \\
& \wedge history' = Append(history, [node \mapsto n, term \mapsto r.term]) \\
& \wedge SendResponse(n, [\\
& \quad type \mapsto WriteResponse, \\
& \quad status \mapsto Ok]) \\
& \vee \wedge \vee election[n] \neq r.election_id \\
& \quad \vee MasterId(election) \neq n \\
& \quad \vee r.token = 0 \\
& \quad \vee r.token < writeTerm \\
& \wedge SendResponse(n, [\\
& \quad type \mapsto WriteResponse, \\
& \quad status \mapsto PermissionDenied]) \\
& \wedge UNCHANGED \langle writeTerm, history \rangle \\
& \wedge DiscardRequest(n) \\
& \wedge UNCHANGED \langle stateVars, election, streamVars \rangle
\end{aligned}$$

\ * Modification History
\ * Last modified Tue Feb 26 13:24:20 PST 2019 by jordanhalterman
\ * Created Wed Feb 20 23:49:17 PST 2019 by jordanhalterman