#include "mac\_dot11\_powersave\_manager.h"

#include "mac\_dot11-sta.h"

#include "phy\_802\_11.h"

using namespace Dot11;

using namespace Qos;

/// \brief Implementation of getMode Function

///

/// This function will return the mode of operation of smps

///

/// \param dot11 dot11 structure pointer

/// \param node node pointer

SmMode SmPowerSave::getMode(Node\* node,

 MacDataDot11\* dot11)

{

 SmMode val = k\_Disabled;

 char smpsAp = k\_Disabled;

 unsigned int phyIndex = (unsigned)dot11->myMacData->phyNumber;

 if (dot11->isHTEnable)

 {

 if (!dot11->associatedAP || PHY\_GetNumConfigAntennas(node, phyIndex) == 1)

 {

 return val;

 }

 smpsAp = dot11->associatedAP->staHtCapabilityElement.

 htCapabilitiesInfo.smPowerSave;

 if (dot11->smpsMode == k\_Static && smpsAp == k\_Static)

 {

 val = k\_Static;

 }

 else if (dot11->smpsMode == k\_Dynamic && smpsAp == k\_Dynamic)

 {

 val = k\_Dynamic;

 }

 return val;

 }

 else

 {

 return val;

 }

}

/// \brief Implementation of getCurrentRfChainMode Function

///

/// This function will return the current rf chain mode

///

/// \param dot11 dot11 structure pointer

/// \param node node pointer

RfChainMode SmPowerSave::getCurrentRfChainMode(Node\* node,

 MacDataDot11\* dot11)

{

 RfChainMode currentMode;

 unsigned int phyIndex = (unsigned)dot11->myMacData->phyNumber;

 int configAntennas = PHY\_GetNumConfigAntennas(node, phyIndex);

 int activeAntennas = PHY\_GetNumActiveAntennas(node, phyIndex);

 if (configAntennas == activeAntennas)

 {

 currentMode = k\_All\_Rf\_Chain;

 }

 else

 {

 currentMode = k\_Single\_Rf\_Chain;

 }

 return currentMode;

}

/// \brief Implementation of switchRfChains Function

///

/// This function switches the number of antenna elements depending

/// on the mode of operation of smps

///

/// \param dot11 dot11 structure pointer

/// \param node node pointer

/// \param mode RfChainMode enum

void Dot11nController::switchRfChains(MacDataDot11\* dot11,

 Node\* node,

 RfChainMode mode)

{

 if (!MacDot11IsAp(dot11))

 {

 SmMode psVal = smps->getMode(node, dot11);

 RfChainMode currentMode = smps->getCurrentRfChainMode(node, dot11);

 switch (psVal)

 {

 case k\_Static :

 if (mode == k\_Single\_Rf\_Chain && currentMode == k\_All\_Rf\_Chain)

 {

 Phy802\_11SetNumActiveAtnaElems(

 node->phyData[dot11->myMacData->phyNumber],

 mode);

 }

 break;

 case k\_Dynamic :

 if (mode != currentMode)

 {

 Phy802\_11SetNumActiveAtnaElems(

 node->phyData[dot11->myMacData->phyNumber],

 mode);

 }

 break;

 default :

 break;

 }

 }

}

/// \brief Implementation of switchRfChains Function

///

/// This function switches the number of antenna elements depending

/// on the mode of operation of smps

///

/// \param dot11 dot11 structure pointer

/// \param node node pointer

/// \param mode RfChainMode enum

void Dot11acController::switchRfChains(MacDataDot11\* dot11,

 Node\* node,

 RfChainMode mode)

{

 if (!MacDot11IsAp(dot11))

 {

 SmMode psVal = smps->getMode(node, dot11);

 RfChainMode currentMode = smps->getCurrentRfChainMode(node, dot11);

 switch (psVal)

 {

 case k\_Static :

 if (mode == k\_Single\_Rf\_Chain && currentMode == k\_All\_Rf\_Chain)

 {

 Phy802\_11SetNumActiveAtnaElems(

 node->phyData[dot11->myMacData->phyNumber],

 mode);

 }

 break;

 case k\_Dynamic :

 if (mode != currentMode)

 {

 Phy802\_11SetNumActiveAtnaElems(

 node->phyData[dot11->myMacData->phyNumber],

 mode);

 }

 break;

 default :

 break;

 }

 }

}

/// \brief Implementation of sleep Function

///

/// This function will make the node to enter in sleep mode

///

/// \param dot11 dot11 structure pointer

/// \param node node pointer

void Dot11acController::sleep(MacDataDot11\* dot11, Node\* node)

{

 if (vhtps->canSleep(dot11, node))

 {

 vhtps->stopListening(dot11, node);

 }

}

/// \brief Implementation of wakeUp Function

///

/// This function will make the node to enter in wake up mode

///

/// \param dot11 dot11 structure pointer

/// \param node node pointer

void Dot11acController::wakeUp(MacDataDot11\* dot11, Node\* node)

{

 vhtps->startListening(dot11, node);

}

/// \brief Implementation of update TxVector Function

///

/// This function will set some values in txVector

///

/// \param dot11 dot11 structure pointer

/// \param tempHextHopAddress address of next hop

/// \param txVector txVector pointer

/// \param node node pointer

void VhtPowerSave::updateTxVector(MacDataDot11\* dot11,

 Mac802Address tempNextHopAddress,

 MAC\_PHY\_TxRxVector\* txVector,

 Node\* node)

{

 if (isModeEnabled(dot11, tempNextHopAddress, node))

 {

 // For future use

 // txVector->txop\_ps\_not\_allowed = FALSE;

 }

}

/// \brief Implementation of can sleep Function

///

/// This function will check whether the station can sleep or not

///

/// \param dot11 dot11 structure pointer

/// \param node node pointer

BOOL VhtPowerSave::canSleep(MacDataDot11\* dot11, Node\* node)

{

 MAC\_PHY\_TxRxVector txVector;

 if (dot11->isVHTEnable)

 {

 if (MacDot11IsAp(dot11))

 {

 return FALSE;

 }

 else

 {

 if (!dot11->associatedAP)

 {

 return FALSE;

 }

 else

 {

 if ((dot11->isVHTTxopPSEnable) &&

 (dot11->associatedAP->vhtInfo.staVhtCapabilityElement.

 m\_capabilitiesInfo.m\_vhtTxopPs))

 {

 // For future use

 /\* PHY\_GetTxVector(node,

 dot11->myMacData->phyNumber,

 txVector);

 if (!txVector.txop\_ps\_not\_allowed)

 {

 return TRUE;

 } \*/

 return TRUE;

 }

 else

 {

 return FALSE;

 }

 }

 }

 }

 else

 {

 return FALSE;

 }

}

/// \brief Implementation of isModeEnabled Function

///

/// This function will check whether the mode is enabled on both the Ap and station

///

/// \param dot11 dot11 structure pointer

/// \param tempNextHopAddress address of next Hop

/// \param node node pointer

BOOL VhtPowerSave::isModeEnabled(MacDataDot11\* dot11,

 Mac802Address tempNextHopAddress,

 Node\* node)

{

 BOOL vhtTxopSta = FALSE;

 if (MacDot11IsAp(dot11) && dot11->isVHTEnable)

 {

 DOT11\_ApStationListItem\* stationItem = NULL;

 stationItem = MacDot11ApStationListGetItemWithGivenAddress(

 node,

 dot11,

 tempNextHopAddress);

 if (!stationItem || !stationItem->data->isVHTEnabledSta)

 {

 return FALSE;

 }

 vhtTxopSta = stationItem->data->vhtInfo.staVhtCapabilityElement.

 m\_capabilitiesInfo.m\_vhtTxopPs;

 if ((dot11->isVHTTxopPSEnable) && (vhtTxopSta))

 {

 return TRUE;

 }

 else

 {

 return FALSE;

 }

 }

 else

 {

 return FALSE;

 }

}

/// \brief Implementation of stop listening Function

///

/// This function will make the radio off

///

/// \param dot11 dot11 structure pointer

/// \param node node pointer

void VhtPowerSave::stopListening(MacDataDot11\* dot11, Node\* node)

{

 MacDot11StationStopListening(node, dot11);

}

/// \brief Implementation of start listening Function

///

/// This function will make the radio on

///

/// \param dot11 dot11 structure pointer

/// \param node node pointer

void VhtPowerSave::startListening(MacDataDot11\* dot11, Node\* node)

{

 MacDot11StationStartListening(node, dot11);

}

/// \brief Implementation of dot11 update txVector Function

///

/// This function will update the txVector

///

/// \param dot11 dot11 structure pointer

/// \param tempNextHopAddress address of next hop

/// \param txVector txVector pointer

/// \param node node pointer

void Dot11acController::dot11\_UpdateTxVector(

 MacDataDot11\* dot11,

 Mac802Address tempNextHopAddress,

 MAC\_PHY\_TxRxVector\* txVector,

 Node\* node)

{

 vhtps->updateTxVector(dot11, tempNextHopAddress, txVector, node);

}