```c
/****************************************************************************
 Module
   TX_Service.c

 Revision
   1.0.1

 Description
   This implements transmission of data using audio

 Notes

 History
 When           Who     What/Why
 -------------- ---     --------
 05/19/20       LG       testing queuing ability and making final conversions
 05/18/20       LG       adding queuing ability
 05/06/20 08:22 hbf      began conversion for Lab 10
 01/16/12 09:58 jec      began conversion from TemplateFSM.c
****************************************************************************/
/*----------------------------- Include Files -----------------------------*/
/* include header files for this state machine as well as any machines at the
   next lower level in the hierarchy that are sub-machines to this machine
*/
#include "ES_Configure.h"
#include "ES_Framework.h"
#include "ES_Port.h"
#include "TX_Service.h"
#include "RX_Service.h"
#include "DAC_Service.h"
#include "SwitchService.h"
#include "LED_Service.h"
#include "dbprintf.h"
#include "GameplayFSM.h"
#include "EventCheckers.h"
#include "inttypes.h"
#include "DataParserService.h"
#include "GameplayFSM.h"

/*----------------------------- Module Defines ----------------------------*/

/*----------------------------- Module Defines ----------------------------*/
#define BIT_TIME 10
#define START_BIT (BIT1HI | BIT0HI)
#define NUM_DATA_BITS 10

/*---------------------------- Module Functions ---------------------------*/
static void StartTransmit(uint8_t Source);
static void StopTransmit(void);
static void SendNextDibit(void);
/*---------------------------- Module Variables ---------------------------*/
// with the introduction of Gen2, we need a module level Priority variable
static uint8_t MyPriority;
static TX_State_t CurrentState;
static uint32_t TX;
static uint8_t TX_Counter;
static uint8_t CheckSum;
static uint16_t Sum;
static const uint8_t MAX_QUEUE_SIZE = 5;
static uint32_t CONNTransmitQueue[6] = {0, 0, 0, 0, 0, 0}; // transmit queue declarations
static uint32_t SONARTransmitQueue[6] = {0, 0, 0, 0, 0, 0};
static uint32_t TORPEDOTransmitQueue[6] = {0, 0, 0, 0, 0, 0};
static uint8_t OriginStation;
static bool JustDied = true;

static uint8_t CONNQueueIndex = 0;
static uint8_t SONARQueueIndex = 0;
static uint8_t TORPEDOQueueIndex = 0;
static struct GameStatus *gamestatus;

/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
    InitTXService

 Parameters
     uint8_t : the priorty of this service

 Returns
     bool, false if error in initialization, true otherwise

 Description
     Saves away the priority, and does any
     other required initialization for this service
 Notes

 Author
     H. Francis, 05/08/20, 15:34
****************************************************************************/
bool InitTXService(uint8_t Priority)
{
  ES_Event_t ThisEvent;

  MyPriority = Priority;
  CurrentState = TX_Initializing;
  gamestatus = getPointerToGameStatus(); // get pointer to be able to access game status struct
  ThisEvent.EventType = ES_INIT; // post initial transition event
  if (ES_PostToService(MyPriority, ThisEvent) == true)
  {
    return true;
  }
  else
  {
    return false;
  }
}

/****************************************************************************
 Function
    PostTXService

 Parameters
     EF_Event_t ThisEvent ,the event to post to the queue

 Returns
     bool false if the Enqueue operation failed, true otherwise

 Description
     Posts an event to this state machine's queue
 Notes

 Author
     H. Francis, 05/08/20, 15:38
****************************************************************************/
bool PostTXService(ES_Event_t ThisEvent)
{
  return ES_PostToService(MyPriority, ThisEvent);
}

/****************************************************************************
 Function
    RunTXService

 Parameters
   ES_Event_t : the event to process

 Returns
   ES_Event, ES_NO_EVENT if no error ES_ERROR otherwise

 Description
   Run state machine for this service
 Notes

 Author
   H. Francis, 05/08/20, 15:40
****************************************************************************/
ES_Event_t RunTXService(ES_Event_t ThisEvent)
{
  ES_Event_t ReturnEvent;
  ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

  switch(CurrentState)
  {
      case TX_Initializing:
      {
          if(ThisEvent.EventType == ES_INIT) CurrentState = TX_Idle; // if the event is ES_INIT go to TX_Idle state
      }
      break;
      case TX_Idle:
      {
          if(ThisEvent.EventType == ES_START_TRANSMIT) // if the event is ES_START_TRANSMIT
          {
              StartTransmit(ThisEvent.EventParam); // start the transmission with the source station sent
              CurrentState = TX_Transmitting; // transition to TX_Transmitting
          }
      }
      break;
      case TX_Transmitting:
      {
          if(ThisEvent.EventType == ES_EOT) // if the event is ES_EOT
          {
              StopTransmit(); // end the transmission
              CurrentState = TX_Idle; // reset to idle mode
              ES_Event_t NewEvent;
              NewEvent.EventType = ES_EOT; // post ES_EOT to GameplayFSM
              PostGameplayFSM(NewEvent);
          }

          if(ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam == TX_TIMER) // if there's an ES_TIMEOUT of the TX_TIMER
          {
              SendNextDibit();
          }
      }
      break;
  }

  return ReturnEvent;
}
/****************************************************************************
 Function
    GetTXState

 Parameters
   none

 Returns
   TX_State_t current state of this function

 Description
   Getter function for this service's state
 Notes

 Author
   H. Francis, 05/08/20, 16:45
****************************************************************************/
TX_State_t GetTXState(void)
{
    return CurrentState;
}
/****************************************************************************
 Function
    StartTransmit

 Parameters
 Source (station where the function is called from)

 Returns
   none

 Description
   Starts transmit and queues up transmission for this slot

 Notes

 Author
   L. Gardner, 05/19/20
****************************************************************************/
static void StartTransmit(uint8_t Source)
{
    CurrentGameStatus_t status = QueryCurrentGameStatus(); // get the current status of play from Gameplay
    uint8_t default_action;
    if(status == Play) { // initialize default actions based on what status of play the game is in
        default_action = NEXT_TURN;
    } else if (status == Dead) {
        default_action = NO_ACTION;
    } else if (status == Sync) {
        default_action = SYNC;
    }
    if (!gamestatus->WeAreAlive) { // if we have died, our default action becomes RIP
        default_action = RIP;
    }
    if (Source == CONN) { // if the source station is CONN
        if((CONNQueueIndex == 0) || (status == Sync) || (status == Dead) || ((!gamestatus->WeAreAlive) && !(JustDied))) { // if the queue is empty, we are in dead time or sync, or we have been dead for awhile
            TX = ConvertToPacket(ENEMY, CONN, default_action); // create a data packet based on the default action
            gamestatus->NumMoveOnsInARow++; //increase the number of default actions in a row
        } else {
            TX = CONNTransmitQueue[0]; // dequeue the next desired action
            for(int i = 1; i < MAX_QUEUE_SIZE+1; i++) { // shift values in the TX queue
                CONNTransmitQueue[i-1] = CONNTransmitQueue[i];
            }
            CONNQueueIndex--; // decrease the queue index value
            gamestatus->NumMoveOnsInARow = 0; // reset the counter of default actions
            gamestatus->onlyHeardNextTurns = false; // set false that we have only heard NEXT_TURNS
            if((JustDied == true) && (!(gamestatus->WeAreAlive))) JustDied = false; // we have been dead for awhile
        }
    }
    else if (Source == SONAR) { // else if the current station is Sonar
        if((SONARQueueIndex == 0) || (status == Sync) || (status == Dead) || (!gamestatus->WeAreAlive)) { // if we need a default action, convert it to a data packet
            TX = ConvertToPacket(ENEMY, SONAR, default_action);
            gamestatus->NumMoveOnsInARow++;
        } else { // otherwise, if we have something in the queue, get ready to transmit it
            TX = SONARTransmitQueue[0]; // dequeue the next desired action
            for(int i = 1; i < MAX_QUEUE_SIZE+1; i++) {
                SONARTransmitQueue[i-1] = SONARTransmitQueue[i]; // shift values in the queue since one has been removed
            }
            SONARQueueIndex--; // decrease the queue index
            gamestatus->NumMoveOnsInARow = 0; // reset the number of default actions in a row for this round
            gamestatus->onlyHeardNextTurns = false; // set false that we have only encountered NEXT_TURNS
        }
    }
    else if (Source == TORPEDO) { // else if current station is TORPEDO
        if((TORPEDOQueueIndex == 0) || (status == Sync) || (status == Dead) || (!gamestatus->WeAreAlive)) { // if we need a default action, convert it to a data packet
            TX = ConvertToPacket(ENEMY, TORPEDO, default_action);
            gamestatus->NumMoveOnsInARow++;
        } else { // otherwise, if we have something in the queue, get ready to transmit it
            TX = TORPEDOTransmitQueue[0]; // dequeue the next desired action
            for(int i = 1; i < MAX_QUEUE_SIZE+1; i++) {
                TORPEDOTransmitQueue[i-1] = TORPEDOTransmitQueue[i]; // shift values in the queue since one has been removed
            }
            TORPEDOQueueIndex--; // decrement the queue index
            gamestatus->onlyHeardNextTurns = false; // set false that we have only encountered NEXT_TURNS
            gamestatus->NumMoveOnsInARow = 0; // reset the number of default actions in a row for this round
        }
    }
    EnableDAC(); // start DAC module
    SendDibit(START_BIT); // send a start bit
    ES_Timer_InitTimer(TX_TIMER, BIT_TIME); // start bit timer
    Sum = 0; // reset Sum value
    TX_Counter = 0; // reset TX_Counter

}

/****************************************************************************
 Function
    SendNextBit

 Parameters
   none

 Returns
   none

 Description
   Sends next dibit of transmission

 Notes

 Author
   L. Gardner, 05/19/20
****************************************************************************/
static void SendNextDibit(void)
{
    if(TX_Counter == (NUM_DATA_BITS + 1)) { // if we have reached the end of the data (and checksum dibits)
        ES_Event_t PostEvent;
        PostEvent.EventType = ES_EOT; // post ES_EOT event to TX and RX Services
        PostTXService(PostEvent);
        PostRXService(PostEvent);
        return;
    }
    uint8_t nextDibit;

    if (TX_Counter < NUM_DATA_BITS) {
        nextDibit = (TX & (BIT30HI | BIT31HI)) >> 30; // shift the nextDibit from MSB to LSB
        Sum += nextDibit; // add to the sum that will be used to calc CheckSum of message
        TX = (TX<<2); // shift the next dibit into the two MSBs in TX
    }
    else if (TX_Counter == NUM_DATA_BITS) { // if we are on the CheckSum calculation
        Sum &= (BIT1HI | BIT0HI); // take the sum's 2 LSBs
        CheckSum = 3-Sum; // subtract the sum from 3 to get the CheckSum
        nextDibit = CheckSum; // set the last dibit to be the calculated CheckSum
    }
    SendDibit(nextDibit); // send the next dibit
    ES_Timer_InitTimer(TX_TIMER, BIT_TIME); // restart the timer
    TX_Counter++;  // increment the TX_Counter
}
/****************************************************************************
 Function
    StopTransmit

 Parameters
   none

 Returns
   none

 Description
   Stops transmit, disables DAC

 Notes

 Author
   H. Francis, 05/08/20, 17:15
****************************************************************************/
static void StopTransmit(void)
{
    DisableDAC();
}
/****************************************************************************
 Function
 TransmitEnqueue

 Parameters
 Source and Message

 Returns
   none

 Description
 Queues up a new transmit

 Notes

 Author
 L. Gardner, 05/18/2020
****************************************************************************/
void TransmitEnqueue(uint8_t SourceStation, uint32_t Message) {
    if (SourceStation == CONN) { // if source station is Conn, queue the message accordingly and increment the index
        if (CONNQueueIndex != MAX_QUEUE_SIZE) {
            CONNTransmitQueue[CONNQueueIndex] = Message;
            CONNQueueIndex++;

        }
    }
    else if (SourceStation == SONAR) { // if source station is Sonar, queue the message accordingly and increment the index
        if (SONARQueueIndex != MAX_QUEUE_SIZE) {
            SONARTransmitQueue[SONARQueueIndex] = Message;
            SONARQueueIndex++;
        }
    }
    else if (SourceStation == TORPEDO) { // if source station is Torpedo, queue the message accordingly and increment the index
        if (TORPEDOQueueIndex != MAX_QUEUE_SIZE) {
            TORPEDOTransmitQueue[TORPEDOQueueIndex] = Message;
            TORPEDOQueueIndex++;
        }
    }
    return;
}

/*------------------------------- Footnotes -------------------------------*/
/*------------------------------ End of file ------------------------------*/
```