```c
/********************************************************************************
 *
 *    ________  _________  ____    ____  _____          ________   ______
 *   |_   __  ||  _   _  ||_   \  /   _||_   _|        |_   __  |.' ____ \
 *     | |_ \_||_/ | | \_|  |   \/   |    | |     ______ | |_ \_|| (___ \_|
 *     |  _| _     | |      | |\  /| |    | |   _|______||  _| _  _.____`.
 *    _| |__/ |   _| |_    _| |_\/_| |_  _| |__/ |     _| |__/ || \____) |
 *   |________|  |_____|  |_____||_____||________|    |________| \______.'
 *
 ********************************************************************************
 *
 * File          : esp.c
 * Version       : 1.0
 *
 ********************************************************************************
 *
 * Description   : Managing ESP32 state machine and commands
 *
 ********************************************************************************
 *
 * Author        : Miguel Santos
 * Date          : 25.09.2023
 *
 ********************************************************************************
 *
 * MPLAB X       : 5.45
 * XC32          : 2.50
 * Harmony       : 2.06
 *
 *******************************************************************************/

#include "esp.h"

/*******************************************************************************/

/* Enable or disable debug of specific parts by (un)comment */
#ifndef DEBUG_LED
    #define DEBUG_LED PORTGbits.RG9
#endif
//#define DEBUG_ESP_UART
//#define DEBUG_ESP_GET
//#define DEBUG_ESP_SEND

/*******************************************************************************/

/* Define UART and interupts used by ESP32 */
#define ESP_USART_ID                  USART_ID_1
#define ESP_INT_SOURCE_USART_ERROR    INT_SOURCE_USART_1_ERROR
#define ESP_INT_SOURCE_USART_RECEIVE  INT_SOURCE_USART_1_RECEIVE
#define ESP_INT_SOURCE_USART_TRANSMIT INT_SOURCE_USART_1_TRANSMIT

/*******************************************************************************/

/* Time to wait for fifo to be filled */
#define ESP_COUNT_RECEIVE_MS 20

/* Time waiting for a response of main app */
#define ESP_COUNT_WAIT_MS 2000

/*******************************************************************************/

/* Declaration of global application data */
ESP_DATA espData;

/*******************************************************************************/

/**
 * @brief ESP_Initialize
 *
 * Initialize ESP32 state machine, counters and FIFOs
 *
 * @param  void
 * @return void
 */
```

```c
void ESP_Initialize ( void )
{
    /* Place the App state machine in its default state. */
    espData.state = ESP_STATE_IDLE;

    /* Initial flags values */
    espData.transmit = false;
    espData.receive  = false;

    /* Initialize timing counters */
    CNT_Initialize(&espData.cntReceive, ESP_COUNT_RECEIVE_MS);
    CNT_Initialize(&espData.cntWait, ESP_COUNT_WAIT_MS);

    /* Initialize FIFO descriptors */
    FIFO_Initialize(&espData.fifoDesc_tx, ESP_FIFO_SIZE,
                    espData.fifoBuff_tx, 0x00);
    FIFO_Initialize(&espData.fifoDesc_rx, ESP_FIFO_SIZE,
                    espData.fifoBuff_rx, 0x00);
}

/******************************************************************************/

/**
 * @brief ESP_Tasks
 *
 * Execute ESP32 state machine, should be called cyclically
 *
 * @param  void
 * @return void
 */
void ESP_Tasks ( void )
{
    /* Check current state. */
    switch ( espData.state )
    {
        /* Waiting for next state */
        case ESP_STATE_IDLE:
        {
            /* Something to transmit in FIFO */
            if(espData.transmit)
            {
                espData.state = ESP_STATE_TRANSMIT;
            }
            /* Something to receive in FIFO */
            else if(espData.receive)
            {
                CNT_Reset(&espData.cntReceive);
                espData.state = ESP_STATE_RECEIVE;
            }
            break;
        }

        /* A new message has to be transmitted */
        case ESP_STATE_TRANSMIT:
        {
            /* Check if data still available in FIFO */
            if(FIFO_GetReadSpace(&espData.fifoDesc_tx))
            {
                /* Enable the interrupt if needed */
                if(!SYS_INT_SourceIsEnabled(ESP_INT_SOURCE_USART_TRANSMIT))
                {
                    SYS_INT_SourceEnable(ESP_INT_SOURCE_USART_TRANSMIT);
                }
            }
            else
            {
                /* Leave state when fifo is empty */
                espData.transmit = false;
                espData.state = ESP_STATE_IDLE;
            }
            break;
        }

        /* Receiving a message by UART */
        case ESP_STATE_RECEIVE:
        {
            /* Waiting for fifo to be filled */
            if(CNT_Check(&espData.cntReceive))
            {
                if(FIFO_GetBuffer(&espData.fifoDesc_rx, (uint8_t*)&espData.resBuffer))
                {
                    /* Get first line of message */
                    espData.p_resBuffer = strtok(espData.resBuffer, "\r\n");

                    /* Flag state */
                    espData.translate = true;

                    /* Change directly to translate state */
                    espData.state = ESP_STATE_TRANSLATE;
                }
                else
                {
                    /* An error occured, going back to IDLE */
                    espData.state = ESP_STATE_IDLE;
                }

                /* Flag state */
                espData.receive = false;
            }
            break;
        }

        /* Translating different parts of the message received */
        case ESP_STATE_TRANSLATE:
        {
            /* A command is detected */
            if(espData.p_resBuffer[0] == 'A' && espData.p_resBuffer[1] == 'T')
            {
                strcpy(espData.atResponse.command, espData.p_resBuffer);
            }
            /* Acknowledge detected */
            if(strcmp(espData.p_resBuffer, AT_ACK_OK) ||
                    strcmp(espData.p_resBuffer, AT_ACK_ERROR))
            {
                strcpy(espData.atResponse.ack, espData.p_resBuffer);
            }
            /* Data is detected */
            else
            {
                strcpy(espData.atResponse.data, espData.p_resBuffer);
            }

            /* Get next line in string */
            espData.p_resBuffer = strtok(NULL, "\r\n");

            /* Leave state when no more lines */
            if(espData.p_resBuffer == NULL)
            {
                /* Reset buffer */
                memset(espData.resBuffer, 0x00, sizeof(espData.resBuffer));

                /* Machine states and flags */
                espData.newMessage = true;
                espData.translate = false;
                espData.wait = true;
                espData.state = ESP_STATE_WAIT;

                CNT_Reset(&espData.cntWait);
            }
            break;
        }

        /* Waiting for main application to answer */
        case ESP_STATE_WAIT:
        {
            if(CNT_Check(&espData.cntWait))
```

```c
                {
                    espData.newMessage = false;
                    espData.state = ESP_STATE_IDLE;
                }
                break;
            }

            /* The default state should never be executed. */
            default:
            {
                /* TODO: Handle error in application's state machine. */
                break;
            }
        }
    }

    /*******************************************************************************/

    /**
     * @brief ESP_SendCommand
     *
     * Send a command to the ESP32, managed by state machine
     *
     * @param  char Command to send ; Use constant definitions
     * @return bool True = command send ; False = Not allowed to send a command
     */
    bool ESP_SendCommand( char *p_command )
    {
        /* Local variables */
        S_Fifo *p_fifoDesc;
        uint8_t commandSize;
        uint8_t i_string;
        bool commandStatus;

        /* DEBUG */
        #ifdef DEBUG_ESP_SEND
            DEBUG_LED = true;
        #endif

        /* Default command status */
        commandStatus = false;

        /* Dont send a command if not IDLE */
        if(espData.state == ESP_STATE_IDLE)
        {
            /* Point to the desired FIFO */
            p_fifoDesc = &espData.fifoDesc_tx;

            /* Get number of characters to send */
            commandSize = strlen(p_command);

            /* Check if enough space in FIFO */
            if(FIFO_GetWriteSpace(p_fifoDesc) >= (commandSize + 2))
            {
                /* Loop to add command */
                for(i_string = 0; i_string < commandSize; i_string++)
                {
                    FIFO_Add(p_fifoDesc,(uint8_t)(p_command[i_string]));
                }

                /* Add CR and LF suffix to FIFO */
                FIFO_Add(p_fifoDesc,(uint8_t)('\r'));
                FIFO_Add(p_fifoDesc,(uint8_t)('\n'));

                /* Command added to FIFO */
                espData.transmit = true;
                commandStatus = true;
            }
        }
        /* DEBUG */
        #ifdef DEBUG_ESP_SEND
            DEBUG_LED = false;
        #endif

        /* Feedback */
        return commandStatus;
    }

    /*******************************************************************************/

    /**
     * @brief _IntHandlerDrvUsartInstance0
     *
     * Interrupt instance to manage UART communication to ESP32
     *
     */
    void __ISR(_UART_1_VECTOR, ipl7AUTO) _IntHandlerDrvUsartInstance0(void)
    {
        S_Fifo *RX_fifoDescriptor;
        S_Fifo *TX_fifoDescriptor;
        uint8_t TX_size;
        uint8_t TX_BufferFull;
        uint8_t dataFifo;
        USART_ERROR usartStatus;

        /* Pointers to fifo descriptors */
        RX_fifoDescriptor = &espData.fifoDesc_rx;
        TX_fifoDescriptor = &espData.fifoDesc_tx;


        /* DEBUG */
        #ifdef DEBUG_ESP_UART
            DEBUG_LED = true;
        #endif

        /* Reading the error interrupt flag */
        if(SYS_INT_SourceStatusGet(ESP_INT_SOURCE_USART_ERROR))
        {
            /* Clear up the error interrupt flag */
            SYS_INT_SourceStatusClear(ESP_INT_SOURCE_USART_ERROR);
        }

        /* Reading the receive interrupt flag */
        if(SYS_INT_SourceStatusGet(ESP_INT_SOURCE_USART_RECEIVE))
        {
            /* Checks overrun or parity error */
            usartStatus = PLIB_USART_ErrorsGet(ESP_USART_ID);

            if(usartStatus)
            {
                /* Errors are auto cleaned when read, except overrun */
                if ( usartStatus & USART_ERROR_RECEIVER_OVERRUN )
                {
                    PLIB_USART_ReceiverOverrunErrorClear(ESP_USART_ID);
                }
            }
            else
            {
                espData.receive = true;

                while(PLIB_USART_ReceiverDataIsAvailable(ESP_USART_ID))
                {
                    dataFifo = PLIB_USART_ReceiverByteReceive(ESP_USART_ID);
                    FIFO_Add(RX_fifoDescriptor, dataFifo);
                }

                /* Clear up the interrupt flag when buffer is empty */
                SYS_INT_SourceStatusClear(ESP_INT_SOURCE_USART_RECEIVE);
            }
        }

        /* Reading the transmit interrupt flag */
        if(SYS_INT_SourceStatusGet(ESP_INT_SOURCE_USART_TRANSMIT))
        {
            TX_size = FIFO_GetReadSpace(TX_fifoDescriptor);
            TX_BufferFull = PLIB_USART_TransmitterBufferIsFull(ESP_USART_ID);

            while(TX_size && !TX_BufferFull)
            {
                FIFO_GetData(TX_fifoDescriptor, &dataFifo);
                PLIB_USART_TransmitterByteSend(ESP_USART_ID, dataFifo);
                TX_size = FIFO_GetReadSpace(TX_fifoDescriptor);
                TX_BufferFull = PLIB_USART_TransmitterBufferIsFull(ESP_USART_ID);
            }

            /* Disable the interrupt, to avoid calling ISR continuously*/
            SYS_INT_SourceDisable(ESP_INT_SOURCE_USART_TRANSMIT);

            /* Clear up the interrupt flag */
            SYS_INT_SourceStatusClear(ESP_INT_SOURCE_USART_TRANSMIT);
        }

        /* DEBUG */
        #ifdef DEBUG_ESP_UART
            DEBUG_LED = false;
        #endif
     }

    /*******************************************************************************/

    /* End of File ****************************************************************/

```