ESP8266 GPS
WIRELESS STRATUM1
NTP SERVER

INTERNET CONNECTED DEVICES AND TIME SYNCHRONIZATION

A lot of devices that are connected to the internet uses NTP protocol to synchronize internal clocks in a periodic way. Personal computers (PC) are the most common case. Those devices usually have an internal clock (RTC), but probably lost a second or more every day. After some days the offset could be from some minutes to hours.

With the near omnipresence of the internet, a lot of connected devices, to lower manufacturing costs, don't include an RTC, so every time a timestamp is needed a call to an NTP server is made.

What if at some point in time, internet connection is lost and hundreds of devices need a timestamp to do some stuff?

How to time synchronize hundreds of wireless connected devices, but in an scenario with no internet connection, i. e. in a faraway forest where there is no cell phone signal?

How to time synchronize industrial devices in factories, where for security and/or paranoia reasons, devices are interconnected but physically unplugged from the internet?

WIRELESS NTP SERVER WITHOUT INTERNET CONNECTION

If an internetless reliable time source is needed, there are multiple choices: Radio time signals, atomic clock, gps and others.

Time signal receivers are a low cost option, but their effectiveness depends on how close are to the emitting station. Also, there are different frequencies depending on the country or geographic zone, so is not a very universal alternative for every site.

Atomic clocks are the best option if a very high degree of precision is needed. Some time ago atomic clocks were big, not very portable and very power hungry. However, with the advances in miniaturization now is possible to get a chip sized atomic clock that doesn't need more than a few milliwatts to work. If cost is not a determinant factor (prices around thousand dollars), this is the option to choose!.

Global Positioning System GPS or equivalent systems in other nations (GLONASS, GALILEO, BEIDOU) are an intermediate option that brings more flexibility than radio time signals (well, some clear sky view is needed) and with a price cost much less than atomic clock.

If a very high degree of robustness is needed, for example to make a homemade nuclear reactor or if big amount of devices doing millions of requests for second will be connected, an already made solution is preferred. But if something simpler is wanted, for experimenting purposes, and offsets of one second could be tolerated, is possible to build one spending around $10 USD using an ESP8266-01 and a GPS module, also a handheld GPS with serial NMEA output could be used.

DATE AND TIME PROTOCOLS: DAYTIME, TIME AND NTP

Since the very first practical computers started to work, the idea of having them time synchronized was revolving around. There are many protocols for this, some of them are: daytime, time and NTP.

Daytime protocol:

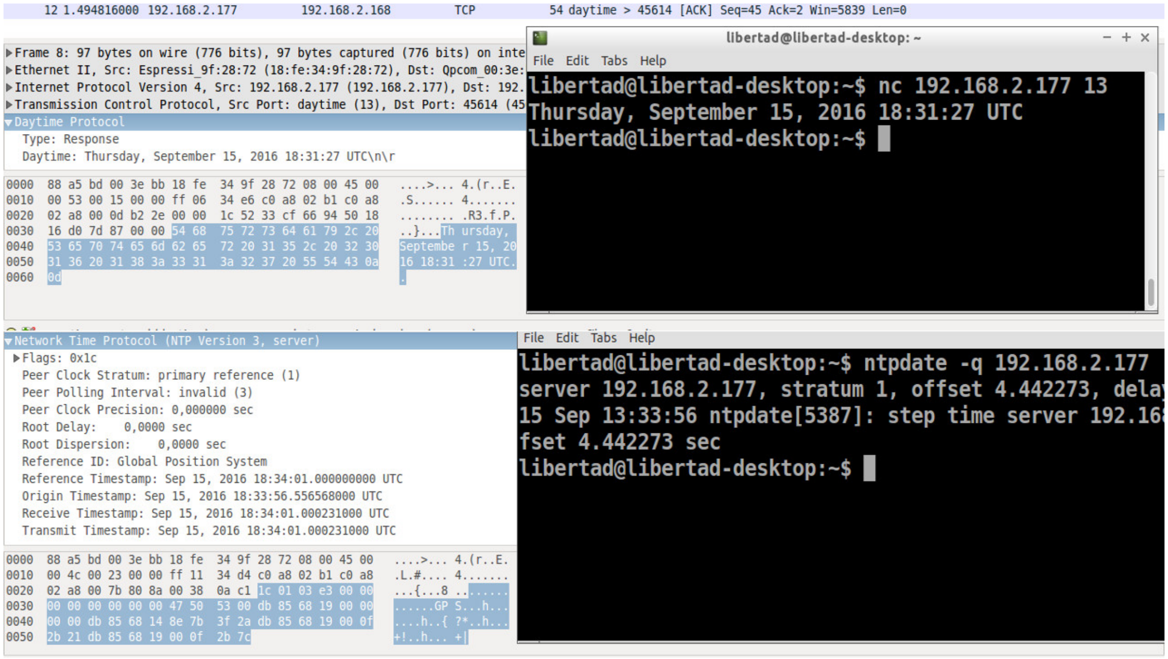
This protocol is described in RFC867. It's one of the oldest and practically not used now. Some time servers still provide it for educational purposes and as an alternative for very very old hardware/software applications that still uses it. This protocol works over port 13 and time/date information is sent in clear text. There is not a specific recommendation for the format used, the only exigence is that the information could be read by a human.

Time protocol:

This protocol is described in RFC868 and works over port 37. Time/date information is coded as seconds elapsed since 00:00:00 (midnight) of January 1 1970 in a 32 bit number.

NTP protocol:

NTP was conceived to bring time/date information, is described in RFC5905 and have a precision in the order of milliseconds. Is based on a modified Marzullo's algorithm to take in count the variable delay of the information packets. This protocol is very widespread used, information packets travel over UDP connections on port 123 to minimize the processing time.



```
   12 1.494816000  192.168.2.177         192.168.2.168         TCP           54 daytime > 45614 [ACK] Seq=45 Ack=2 Win=5839 Len=0

▶Frame 8: 97 bytes on wire (776 bits), 97 bytes captured (776 bits) on inte
▶Ethernet II, Src: Espressi_9f:28:72 (18:fe:34:9f:28:72), Dst: Qpcom_00:3e:
▶Internet Protocol Version 4, Src: 192.168.2.177 (192.168.2.177), Dst: 192.
▶Transmission Control Protocol, Src Port: daytime (13), Dst Port: 45614 (45
▼Daytime Protocol
   Type: Response
   Daytime: Thursday, September 15, 2016 18:31:27 UTC\n\r

0000  88 a5 bd 00 3e bb 18 fe  34 9f 28 72 08 00 45 00   ....>... 4.(r..E.
0010  00 53 00 15 00 00 ff 06  34 e6 c0 a8 02 b1 c0 a8   .S...... 4.......
0020  02 a8 00 0d b2 2e 00 00  1c 52 33 cf 66 94 50 18   ........ .R3.f.P.
0030  16 d0 7d 87 00 00 54 68  75 72 73 64 61 79 2c 20   ..}...Th ursday, 
0040  53 65 70 74 65 6d 62 65  72 20 31 35 2c 20 32 30   Septembe r 15, 20
0050  31 36 20 31 38 3a 33 31  3a 32 37 20 55 54 43 0a   16 18:31 :27 UTC.
0060  0d                                                 .
```

```
libertad@libertad-desktop:~$ nc 192.168.2.177 13
Thursday, September 15, 2016 18:31:27 UTC
libertad@libertad-desktop:~$ 
```

```
▼Network Time Protocol (NTP Version 3, server)
  ▶Flags: 0x1c
   Peer Clock Stratum: primary reference (1)
   Peer Polling Interval: invalid (3)
   Peer Clock Precision: 0,000000 sec
   Root Delay:    0,0000 sec
   Root Dispersion:    0,0000 sec
   Reference ID: Global Position System
   Reference Timestamp: Sep 15, 2016 18:34:01.000000000 UTC
   Origin Timestamp: Sep 15, 2016 18:33:56.556568000 UTC
   Receive Timestamp: Sep 15, 2016 18:34:01.000231000 UTC
   Transmit Timestamp: Sep 15, 2016 18:34:01.000231000 UTC

0000  88 a5 bd 00 3e bb 18 fe  34 9f 28 72 08 00 45 00   ....>... 4.(r..E.
0010  00 4c 00 23 00 00 ff 11  34 d4 c0 a8 02 b1 c0 a8   .L.#.... 4.......
0020  02 a8 00 7b 80 8a 00 38  0a c1 1c 01 03 e3 00 00   ...{...8 ........
0030  00 00 00 00 00 00 47 50  53 00 db 85 68 19 00 00   ......GP S...h...
0040  00 00 db 85 68 14 8e 7b  3f 2a db 85 68 19 00 0f   ....h..{ ?*..h...
0050  2b 21 db 85 68 19 00 0f  2b 7c                     +!..h... +|
```

```
libertad@libertad-desktop:~$ ntpdate -q 192.168.2.177
server 192.168.2.177, stratum 1, offset 4.442273, delay
15 Sep 13:33:56 ntpdate[5387]: step time server 192.168
fset 4.442273 sec
libertad@libertad-desktop:~$ 
```

ESP8266 + GPS

This project started as an improvement of Ray Burnette's Tardis Time that basically has the same hardware elements, but doesn't send time/date information using standard protocols, so a small listener application must be developed in every different device to be synchronized. This project takes advantage of the built-in defacto time synch apps in personal computers, raspberry pi, and could be used by the NTP client libraries for the ESP8266.

The GPS module used was an EM-506 which don't have a PPS signal. Also a handheld GPS receiver with RS232 NMEA output could be adapted for the same purpose. The ESP8266 was programmed using SDK version 1.5.2. Don't forget that some kind of USB to TTL 3V interface is necessary for programming!. The software could be described as 3 big parts:

SERIAL DATA RECEPTION

ESP8266's UART has a hardware FIFO with a max capacity of 256 characters and multiple sources of interruptions. The character received threshold and timeout interrupts were used in the program.

Character threshold interrupt was used to fire automatically a function when some amount of characters reaches the UART's FIFO. In this particular case as NMEA strings are expected from the GPS, and they are around 30-80 chars long, a threshold number between them was used. This function reads characters from the FIFO and writes them to a circular buffer.

Timeout interrupt is used when only a few characters were received (less than threshold amount) and no more characters were detected in a determinate time interval. In this particular case a timeout of 10 byte-time was programmed.

This function need to be executed quickly to return from the interrupt as fast as possible, so no processing is done here, just read-and-copy.

STRING PARSING, ACTUAL TIME/DATE AND MICROSECONDS CORRECTION

Serial ISR writes received characters in a circular buffer which max size should be twice the length of the longest NMEA string to be parsed. In this particular case the string that begins with the $GPRMC header. This particular string holds time/date information. The function that processes strings is called by the ISR once all received characters were written to the circular buffer using the messaging system provided by the SDK system_os_post.

NMEA strings have variable length, so absolute character position should be avoided to parse values. The number of delimiters in this case commas "," is constant for a specific type of string and were used to extract the time and date values. A sample string looks like this:

$GPRMC,201705.000,A,0000.0000,N,00000.0000,W,1.10,265.50,120816,,,A*79

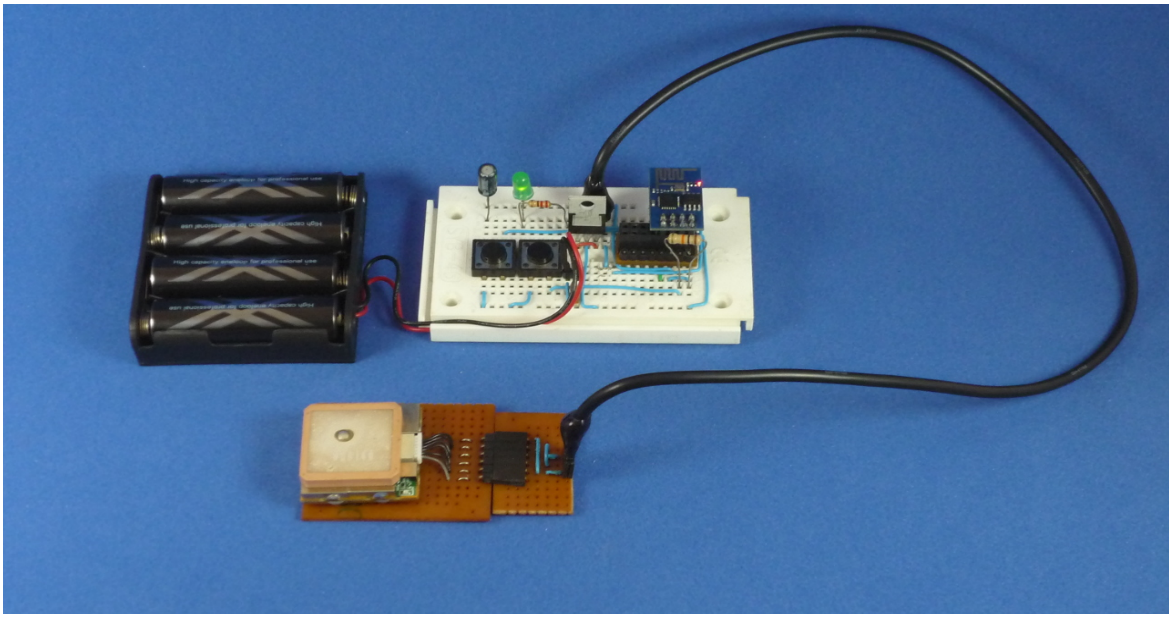
Between comma "," delimiters #1 y #2 lies the string 201705.000 which means 20 hours, 17 minutes and 05,000 seconds. Between comma "," delimiters #9 y #10 lies the string 120816 which means Day 12, month 08 (August) and Year 16 (2016).

Additionally the system's microseconds ring counter (not from GPS) is stored, to make adjustments and deliver a more accurate timestamp.

DELIVER DATA, SOCKETS

Three independent sockets were created to listen to any type of service: daytime, time or NTP. For daytime, when a query is received, data are read from the global variables that holds the date and time information and transformed to a human readable text string. For the time service case, date and time information should be encoded in a suitable way, the helper function SecondsSince1900 do the task.

In the case of NTP, the received packet should be temporarily stored, some information needs to be modified/updates and sent back as an answer. Due to the GPS used in the project only brings time/date information each second, and an NTP query could occur in the middle of that interval, to mitigate a bit this problem, the value of the system microsecond ring counter is read using system_get_time() every time a new GPS frame is received. When an NTP query is received, that counter is read again, and with these two values the time elapsed between the last GPS update and NTP query reception could be estimated. This time is added to the time stored in global variables (the one that is updated every second) and send in the answering NTP packet.



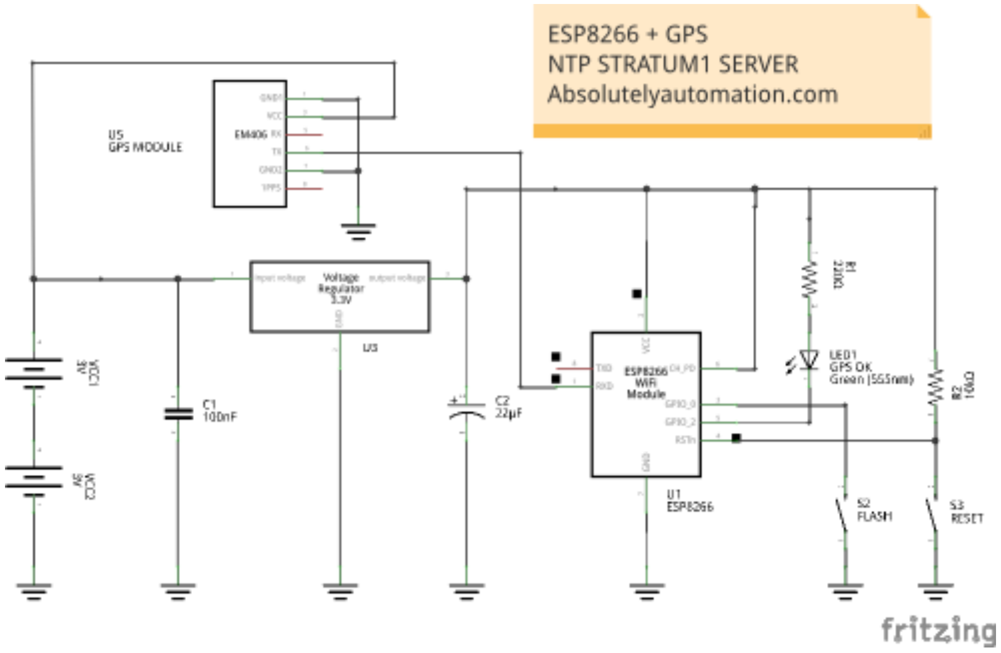
TESTS AND CONCLUSIONS

- Standard apps were used (In this particular case Linux apps) to test time services programmed in the ESP8266, like netca, rdate, ntpdate.
- Due to its small size, low power and wireless connectivity, could be installed in places with better clear view of the sky: near windows, rooftops, etc.
- With a GPS with a PPS, more precision could be achieved.
- Could be solar powered (with batteries for the night) to get a totally wireless continuous operating system.
- Some signal level converted have to be used depending of the type of the GPS like 5V TTL to 3V TTL or RS232 to 3V TTL.

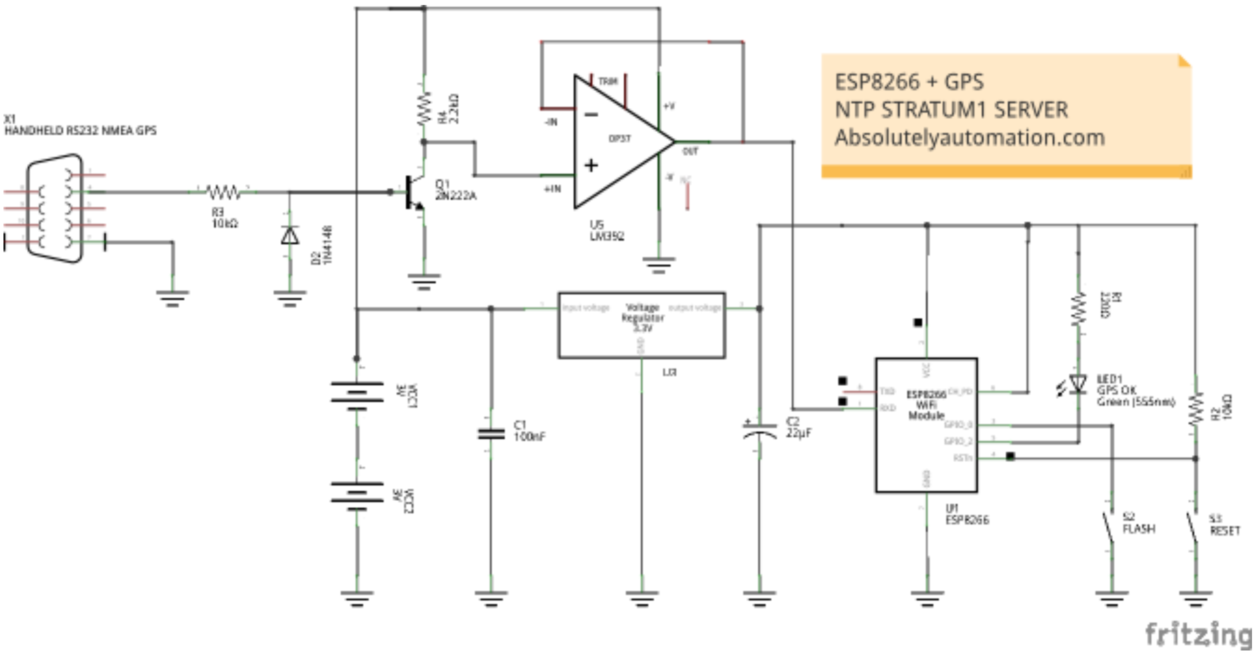
LINKS

Video that shows all the components and working tests : **youtu.be/JWkGUuhKO2E**
ESP8266-01: **http://s.click.aliexpress.com/e/NrBqvnEIY**
GPS module: **http://s.click.aliexpress.com/e/JUjamQB2z**
USB to TTL interface: **http://s.click.aliexpress.com/e/ZrnaURjI6**

GPS MODULE TO ESP8266 INTERFACE SCHEMATIC



HANDHELD GPS GARMIN RECEIVER TO ESP8266 INTERFACE SCHEMATIC



user_config.h

```
#define SSID "YOURAPSSID"
#define SSID_PASSWORD "yourappasswd"
#define HOST_NAME "ESP_NTP_STRATUM1"
```

Makefile

```
# A very very simple makefile copied from the bliky example of SDK 1.5.2
# requires: export PATH=/opt/Expressif/esp-open-sdk/xtensa-lx106-elf/bin:$PATH
# some things need to be manually adjusted for different type and sizes of
flash memory

CC = xtensa-lx106-elf-gcc
CFLAGS = -I. -mlongcalls
LDLIBS = -nostdlib -Wl,--start-group -lmain -lnet80211 -lwpa -llwip -lpp -
lphy -Wl,--end-group -lgcc
LDFLAGS = -Teagle.app.v6.ld

stratum1-0x00000.bin: stratum1
      esptool.py elf2image $^

stratum1: stratum1.o

stratum1.o: stratum1.c

flash: stratum1-0x00000.bin
      esptool.py write_flash 0 stratum1-0x00000.bin 0x40000
stratum1-0x40000.bin

clean:
      rm -f stratum1 stratum1.o stratum1-0x00000.bin stratum1-0x400000.bin
```

stratum1.c

```
/
******************************************************************************
 * Absolutelyautomation.com
 *
 * GPS based Stratum 1 NTP server, also daytime and time services
 *
 * Daytime port 13      rfc867            tcp
 * Time     port 37     rfc868            tcp
 * NTP            port 123    rfc1305     udp
******************************************************************************
*/

#include "ets_sys.h"
#include "osapi.h"
#include "mem.h"
#include "gpio.h"
#include "os_type.h"
```

```
#include "user_interface.h"
#include "espconn.h"
#include <include/driver/uart.h>


const char debug_udp_remote_ip[4] = { 192, 168, 2, 168 };
const unsigned short debug_udp_remote_port = 12345;


#define UART_RX_BUFF_SIZE     150
#define TRANS_QUEUE_LEN       20
#define CONN_QUEUE_LEN        20
#define PROC_TASK_PRIO  0
#define CONN_TASK_PRIO  1
#define TRANS_RECV_DATA_FROM_UART   1
#define CONN_CLOSE_TCP_DAYTIME      2
#define CONN_CLOSE_TCP_TIME   3


#define GPRMC_MAX_SIZE        72
const uint32 GPRMC_US_OFFSET_BYTETIME = GPRMC_MAX_SIZE *2000 ;    // Byte time
@ 4800bps = 2000 us
static os_event_t trans_Queue[TRANS_QUEUE_LEN];
static os_event_t conn_Queue[CONN_QUEUE_LEN];
char NMEA_buffer[UART_RX_BUFF_SIZE] ;


static const int gpioledpin = 2;
LOCAL os_timer_t gps_watchdog;
LOCAL os_timer_t ipready_timer;
LOCAL struct espconn socket_udp_debug;
LOCAL struct espconn socket_tcp_daytime;
LOCAL struct espconn socket_tcp_time;
LOCAL struct espconn socket_udp_ntp;
static struct UartBuffer *pRxBuffer = NULL;

uint8 GPSfix;
uint8 GPS_wd_flag;
uint8 GPSsecond;
uint8 GPSminute;
uint8 GPShour;
uint16 GPSmillisecond;
uint16 GPSyear;
uint8 GPSmonth;
uint8 GPSday;
uint32 RTCcounts;

/
******************************************************************************
 * FunctionName : gps_wd_func
 * Description  : Run by a software timer every 3 seconds, checks if there
is no new valid NMEA frame to turn off GPSfix flag
 *
 * Parameters   : NONE
 * Returns      : NONE

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

void gps_wd_func(void *arg)
{
      // If no serial string received in 3 seconds, set GPS watchdog flag to
0 and GPSfix to 0
      // GPS_wd_flag should be raised every second if a valid GPS NMEA string
is received

      if(GPS_wd_flag == 0)
      {
            GPSfix=0;
      }
      GPS_wd_flag=0;

}

/
******************************************************************************
 * FunctionName : uart0_rx_intr_disable
 * Description  : disables UART0 full rx buffer and rx timeout interrupts
 *
 * Parameters   : NONE
 * Returns      : NONE

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

void uart0_rx_intr_disable(void)
{
    CLEAR_PERI_REG_MASK(UART_INT_ENA(0), UART_RXFIFO_FULL_INT_ENA |
UART_RXFIFO_TOUT_INT_ENA);
}

/
******************************************************************************
 * FunctionName : uart0_rx_intr_enable
 * Description  : enables UART0 full rx buffer and rx timeout interrupts
 *
 * Parameters   : NONE
 * Returns      : NONE

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

void uart0_rx_intr_enable(void)
{
    SET_PERI_REG_MASK(UART_INT_ENA(0), UART_RXFIFO_FULL_INT_ENA |
UART_RXFIFO_TOUT_INT_ENA);
}
```

```
/
******************************************************************************
 * FunctionName : uart0_rx_intr_handler
 * Description  : Internal used function
 *                UART0 interrupt handler, add self handle code inside. !
MODIFIED only RX interrupts !
 * Parameters   : void *para - point to ETS_UART_INTR_ATTACH's arg
 * Returns      : NONE

******************************************************************************
*/
LOCAL void
uart0_rx_intr_handler(void *para)
{

    if (UART_FRM_ERR_INT_ST == (READ_PERI_REG(UART_INT_ST(0)) &
UART_FRM_ERR_INT_ST)) {
        WRITE_PERI_REG(UART_INT_CLR(0), UART_FRM_ERR_INT_CLR);
    }

    if (UART_RXFIFO_FULL_INT_ST == (READ_PERI_REG(UART_INT_ST(0)) &
UART_RXFIFO_FULL_INT_ST)) {
        uart0_rx_intr_disable();
        rx_sliding_buff_enq();
    } else if (UART_RXFIFO_TOUT_INT_ST == (READ_PERI_REG(UART_INT_ST(0)) &
UART_RXFIFO_TOUT_INT_ST)) {
        uart0_rx_intr_disable();
        rx_sliding_buff_enq();
    }

}

/
******************************************************************************
 * FunctionName : SecondsSince1900
 * Description  : Returns input date in seconds since 00:00 GMT JAN 01 1900
 *                                  No range verification! use with care!
 * Parameters   : year > 2015, month (1-12), day (1-31), hour (0-23), minute
(0-59), second (0-59)
 * Returns      : Integer 32 bits

******************************************************************************
*/
uint32 ICACHE_FLASH_ATTR
SecondsSince1900(uint16 year, uint8 month, uint8 day, uint8 hour, uint8
minute, uint8 second)
{
      uint16 counter;
      uint16 yearcount,leapdays;
      uint32 daysfromepoch;
      uint16 daysUpToMonth[12] = { 0, 31, 59, 90, 120, 151, 181, 212, 243,
273, 304, 334 };
      uint16 daysUpToMonthLeapYear[12] = { 0, 31, 60, 91, 121, 152, 182, 213,
244, 274, 305, 335 };
```

```
      // integer years from epoch
      yearcount=year-1900;

      // get leapdays from epoch
      leapdays=0;
      for(counter=1900;counter<year;counter++)
      {
            if( ( ( counter%4==0) && (counter%100!=0)) || (counter%400==0 ) )
            {
                  leapdays++;
            }
      }



      if( ( ( year%4==0) && (year%100!=0)) || (year%400==0 ) )
      {
            daysfromepoch = yearcount*365 + leapdays + 
daysUpToMonthLeapYear[month-1]+(day-1);
      }
      else
      {
            daysfromepoch = yearcount*365 + leapdays + daysUpToMonth
[month-1]+(day-1);
      }

      return( (daysfromepoch*86400)+(hour*3600)+(minute*60)+second );
}

/
******************************************************************************
 * FunctionName : Uart_Buf_Init
 * Description  : tx buffer enqueue: fill a first linked buffer
 * Parameters   : char *pdata - data point  to be enqueue
 * Returns      : NONE

******************************************************************************
*/
struct UartBuffer *ICACHE_FLASH_ATTR
Uart_Buf_Init(uint32 buf_size)
{
    uint32 heap_size = system_get_free_heap_size();

    if(buf_size > 65535) { // now not support
        os_printf("no buf for uart\n\r");
        return NULL;
    }
    if (heap_size <= buf_size) {
        os_printf("no buf for uart\n\r");
        return NULL;
    } else {
        os_printf("test heap size: %d\n\r", heap_size);
        struct UartBuffer *pBuff = (struct UartBuffer *)os_malloc
((uint32)sizeof(struct UartBuffer));
        pBuff->UartBuffSize = (uint16)buf_size; // THIS OK
        pBuff->pUartBuff = (uint8 *)os_malloc(pBuff->UartBuffSize);
        pBuff->pInPos = pBuff->pUartBuff;
        pBuff->pOutPos = pBuff->pUartBuff;
        pBuff->Space = pBuff->UartBuffSize;
        pBuff->BuffState = OK;
        pBuff->nextBuff = NULL;
        //        pBuff->TcpControl = RUN;
        return pBuff;
    }
}

/
******************************************************************************
 * FunctionName : rx_sliding_buff_enq
 * Description  : enables UART0 full rx buffer and rx timeout interrupts
 *
 * Parameters   : NONE
 * Returns      : NONE

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

rx_sliding_buff_enq(void)
{
    uint8 fifo_len = 0;
    uint8 str_len = 0;
    uint8 str_len2 = 0;
    ETSParam par = 0;


    uint8* tail = (pRxBuffer->pUartBuff + pRxBuffer->UartBuffSize);

    fifo_len = (READ_PERI_REG(UART_STATUS(UART0)) >>
UART_RXFIFO_CNT_S)&UART_RXFIFO_CNT;


    while (fifo_len--) {
        *(pRxBuffer->pInPos++) = READ_PERI_REG(UART_FIFO(UART0)) & 0xFF;
        if (pRxBuffer->pInPos == tail) {
            pRxBuffer->pInPos = pRxBuffer->pUartBuff;
        }
    }

        // Arranging characteres in a "sliding window" buffer
      str_len = tail - (pRxBuffer->pInPos);
      os_memcpy(NMEA_buffer, pRxBuffer->pInPos, str_len);
      str_len2 = (pRxBuffer->pInPos)-(pRxBuffer->pUartBuff);
      os_memcpy(NMEA_buffer+str_len, pRxBuffer->pUartBuff, str_len2);

    if(system_os_post(PROC_TASK_PRIO, (ETSSignal)TRANS_RECV_DATA_FROM_UART,
par) != TRUE) {
        os_printf("POST proc data from UART0 fail! !\n\r");
    }
    WRITE_PERI_REG(UART_INT_CLR(0), UART_RXFIFO_FULL_INT_CLR |
UART_RXFIFO_TOUT_INT_CLR);
    uart0_rx_intr_enable();
}

/
******************************************************************************
 * FunctionName : uart0_int_setup
 * Description  : Simpler and flexible function to setup UART0 interrupts
 * Parameters  : interrupt bit, additional parameter
 *                         additional parameter is only used for:
RXFIFO_FULL_INT, RXFIFO_TOUT_INT, TXFIFO_EMPTY_INT
 * Returns      : NONE

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

void ICACHE_FLASH_ATTR

uart0_int_setup(unsigned short intbit,unsigned char parameter)
{
    unsigned long regvalue;

    SET_PERI_REG_MASK(UART_INT_ENA(0), intbit );
    if( intbit == UART_RXFIFO_FULL_INT_ENA )
    {
      regvalue=READ_PERI_REG(UART_CONF1(0)) | ( (100 &
parameter)<<UART_RXFIFO_FULL_THRHD_S);
      WRITE_PERI_REG(UART_CONF1(0),regvalue);
    }

    if( intbit == UART_RXFIFO_TOUT_INT_ENA)
    {
      //SET_PERI_REG_MASK(UART_CONF1(0), UART_RX_TOUT_EN  );
      regvalue=READ_PERI_REG(UART_CONF1(0)) | UART_RX_TOUT_EN | ( (0x02 &
parameter)<<UART_RX_TOUT_THRHD_S);
      WRITE_PERI_REG(UART_CONF1(0),regvalue);
    }

    if( intbit == UART_TXFIFO_EMPTY_INT_ST)
    {
      regvalue=READ_PERI_REG(UART_CONF1(0)) | ( (0x10 &
parameter)<<UART_TXFIFO_EMPTY_THRHD_S);
      WRITE_PERI_REG(UART_CONF1(0),regvalue);
    }
    //clear rx and tx fifo,not ready
   SET_PERI_REG_MASK(UART_CONF0(0), UART_RXFIFO_RST | UART_TXFIFO_RST);    //
RESET FIFO
   CLEAR_PERI_REG_MASK(UART_CONF0(0), UART_RXFIFO_RST | UART_TXFIFO_RST);

}
```

```
/
******************************************************************************
 * FunctionName : tcp_daytime_sent_cb
 * Description  : Call back function after a daytime message (port 13)
delivered, Sends a message to a task for tcp comm closing
 * Parameters  :
 *
 * Returns      : NONE

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

void ICACHE_FLASH_ATTR
tcp_daytime_sent_cb(void *arg)
{
      ETSParam par = 0;
      if(system_os_post(CONN_TASK_PRIO, (ETSSignal)CONN_CLOSE_TCP_DAYTIME,
par) != TRUE)
            {
                  os_printf("Close TCP Daytime fail! !\n\r");
            }
}

/
******************************************************************************
 * FunctionName : tcp_time_sent_cb
 * Description  : Call back function after a time message (port 37)
delivered, Sends a message to a task for tcp comm closing
 * Parameters  :
 *
 * Returns      : NONE

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

void ICACHE_FLASH_ATTR
tcp_time_sent_cb(void *arg)
{
      ETSParam par = 0;
      if(system_os_post(CONN_TASK_PRIO, (ETSSignal)CONN_CLOSE_TCP_TIME,
par) != TRUE)
            {
                  os_printf("Close TCP Time fail! !\n\r");
            }
}

/
******************************************************************************
 * FunctionName : udp_ntp_recvcb
 * Description  : Call back function after a NTP message (port 123)
received, processes the packet and send the answer
 * Parameters  :
 *
 * Returns      : NONE
```

```
******************************************************************************
*/

void ICACHE_FLASH_ATTR
udp_ntp_recvcb(void *arg, char *pdata, unsigned short len)
{
      struct espconn *pesp_conn = arg;
      remot_info *premot = NULL;
      char ntp_packet[48];
      uint32 timetmp;
      uint32 fractmp;
      uint32 useconds;

      if (espconn_get_connection_info(pesp_conn,&premot,0) == ESPCONN_OK)
            {

                  if(len>47)
                  {
                        // read incoming ntp packet and store localy
                        os_memcpy(&ntp_packet[0], pdata,48 );
                        pesp_conn->proto.tcp->remote_port = premot-
>remote_port;
                        pesp_conn->proto.tcp->remote_ip[0] = premot-
>remote_ip[0];
                        pesp_conn->proto.tcp->remote_ip[1] = premot-
>remote_ip[1];
                        pesp_conn->proto.tcp->remote_ip[2] = premot-
>remote_ip[2];
                        pesp_conn->proto.tcp->remote_ip[3] = premot-
>remote_ip[3];

                        //NTP values!!!

                        if(GPSfix==1)
                        {
                              ntp_packet[0]=0x1C;     // Leap 0x0, Version
0x3, Mode 0x4
                        }
                        else
                        {
                              ntp_packet[0]=0xDC;     // Leap 0x3, Version
0x3, Mode 0x4
                        }


                        ntp_packet[1]=0x01;     // Stratum 0x1
                        ntp_packet[2]=0x03;     // Poll  0x3 (invalid)
                        ntp_packet[3]=0xE3;     // Precision  0,00000 sec?

                        ntp_packet[4]=0;        // Root Delay?
                        ntp_packet[5]=0;        // Root Delay?
                        ntp_packet[6]=0;        // Root Delay?
                        ntp_packet[7]=0;        // Root Delay?
                        ntp_packet[8]=0;        // Root Dispersion?
                        ntp_packet[9]=0;        // Root Dispersion?
                        ntp_packet[10]=0; // Root Dispersion?
                        ntp_packet[11]=0; // Root Dispersion?

                        ntp_packet[12]=0x47;    // Reference ID, "G"
                        ntp_packet[13]=0x50;    // Reference ID, "P"
                        ntp_packet[14]=0x53;    // Reference ID, "S"
                        ntp_packet[15]=0x00;    // Reference ID, 0x00

                        timetmp=SecondsSince1900(GPSyear, GPSmonth, GPSday,
GPShour, GPSminute, GPSsecond);

                        ntp_packet[16]=(char)(0x000000FF & timetmp >> 24
);    // Reference timestamp seconds
                        ntp_packet[17]=(char)(0x000000FF & timetmp >> 16
);    // Reference timestamp seconds
                        ntp_packet[18]=(char)(0x000000FF & timetmp >> 8 );
// Reference timestamp seconds
                        ntp_packet[19]=(char)(0x000000FF & timetmp);    //
Reference timestamp seconds

                        ntp_packet[20]=0x0;     // Reference timestamp
fraction
                        ntp_packet[21]=0x0;     // Reference timestamp
fraction
                        ntp_packet[22]=0x0;     // Reference timestamp
fraction
                        ntp_packet[23]=0x0;     // Reference timestamp
fraction

                        // Origin timestamp [24]..[27] seconds , [28]..[31]
fraction

                        ntp_packet[24]=ntp_packet[40];
                        ntp_packet[25]=ntp_packet[41];
                        ntp_packet[26]=ntp_packet[42];
                        ntp_packet[27]=ntp_packet[43];

                        ntp_packet[28]=ntp_packet[44];
                        ntp_packet[29]=ntp_packet[45];
                        ntp_packet[30]=ntp_packet[46];
                        ntp_packet[31]=ntp_packet[47];


                        timetmp=SecondsSince1900(GPSyear, GPSmonth, GPSday,
GPShour, GPSminute, GPSsecond);
                        fractmp=system_get_time();

                        if( fractmp >= RTCcounts )
                        {
                              useconds=fractmp - RTCcounts;
                        }
                        else
                        {
                              useconds= (0xFFFFFFFF - RTCcounts)+fractmp;
                        }

                        useconds=useconds+GPRMC_US_OFFSET_BYTETIME;

                        ntp_packet[32]=(char)(0x000000FF & timetmp >> 24
);    // Receive timestamp seconds
                        ntp_packet[33]=(char)(0x000000FF & timetmp >> 16
);    // Receive timestamp seconds
                        ntp_packet[34]=(char)(0x000000FF & timetmp >> 8 );
// Receive timestamp seconds
                        ntp_packet[35]=(char)(0x000000FF & timetmp);    //
Receive timestamp seconds

                        ntp_packet[36]=(char)(0x000000FF & useconds >> 24
);    // Receive timestamp fraction
                        ntp_packet[37]=(char)(0x000000FF & useconds >> 16
);    // Receive timestamp fraction
                        ntp_packet[38]=(char)(0x000000FF & useconds >> 8 );
// Receive timestamp fraction
                        ntp_packet[39]=(char)(0x000000FF & useconds);   //
Receive timestamp fraction

                        timetmp=SecondsSince1900(GPSyear, GPSmonth, GPSday,
GPShour, GPSminute, GPSsecond);
                        fractmp=system_get_time();

                        if( fractmp >= RTCcounts )
                        {
                              useconds=fractmp - RTCcounts;
                        }
                        else
                        {
                              useconds= (0xFFFFFFFF - RTCcounts)+fractmp;
                        }

                        useconds=useconds+GPRMC_US_OFFSET_BYTETIME;

                        ntp_packet[40]=(char)(0x000000FF & timetmp >> 24
);    // Transmit timestamp seconds
                        ntp_packet[41]=(char)(0x000000FF & timetmp >> 16
);    // Transmit timestamp seconds
                        ntp_packet[42]=(char)(0x000000FF & timetmp >> 8 );
// Transmit timestamp seconds
                        ntp_packet[43]=(char)(0x000000FF & timetmp);    //
Transmit timestamp seconds

                        ntp_packet[44]=(char)(0x000000FF & useconds >> 24
);    // Transmit timestamp fraction
                        ntp_packet[45]=(char)(0x000000FF & useconds >> 16
);    // Transmit timestamp fraction
```

```
                        ntp_packet[46]=(char)(0x000000FF & useconds >> 8 );
// Transmit timestamp fraction
                        ntp_packet[47]=(char)(0x000000FF & useconds);   //
Transmit timestamp fraction

                        espconn_sent(pesp_conn, &ntp_packet[0], 48);
                  }


                  //espconn_sent(pesp_conn, pusrdata, os_strlen(pusrdata));
                  //os_printf("UDP data Ntp received %d bytes %d.%d.%d.%
d : %d !!\n\r",len,premot->remote_ip[0],premot->remote_ip[1],premot->remote_ip
[2],premot->remote_ip[3],premot->remote_port);
            }


}

/
******************************************************************************
 * FunctionName : tcp_time_listen
 * Description  : Socket listener function for time service (port 37),
processes the packet and send the answer
 * Parameters   :
 *
 * Returns      : NONE

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

static void ICACHE_FLASH_ATTR
tcp_time_listen(void *arg)
{

      char packet[4];
      uint32 tmptime;

      tmptime=SecondsSince1900
(GPSyear,GPSmonth,GPSday,GPShour,GPSminute,GPSsecond);

      packet[0]=(char)(0x000000FF & tmptime >> 24 );
      packet[1]=(char)(0x000000FF & tmptime >> 16 );
      packet[2]=(char)(0x000000FF & tmptime >> 8 );
      packet[3]=(char)(0x000000FF & tmptime);

      espconn_sent( &socket_tcp_time, &packet[0], 4 );

      //os_printf("Connection on port 37");
      return;
}

/
******************************************************************************
 * FunctionName : tcp_daytime_listen
 * Description  : Socket listener function for daytime service (port 13),
processes the packet and send the answer
 * Parameters  :
 *
 * Returns      : NONE

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

static void ICACHE_FLASH_ATTR
tcp_daytime_listen(void *arg)
{
      char message[60];
      char tmpmesg[60];
      char smonth[20];
      char wday[20];


      switch(GPSmonth)
      {

            case 1  :
                  os_sprintf(&smonth[0],"January");
                  break;
            case 2  :
                        os_sprintf(&smonth[0],"February");
                  break;
            case 3  :
                        os_sprintf(&smonth[0],"March");
                  break;
            case 4  :
                        os_sprintf(&smonth[0],"April");
                  break;
            case 5  :
                        os_sprintf(&smonth[0],"May");
                  break;
            case 6  :
                        os_sprintf(&smonth[0],"June");
                  break;
            case 7  :
                        os_sprintf(&smonth[0],"July");
                  break;
            case 8  :
                        os_sprintf(&smonth[0],"August");
                  break;
            case 9  :
                        os_sprintf(&smonth[0],"September");
                  break;
            case 10  :
                        os_sprintf(&smonth[0],"October");
                  break;
            case 11  :
                        os_sprintf(&smonth[0],"November");
                  break;
            case 12  :
                        os_sprintf(&smonth[0],"December");
                  break;


      }

      // Tomohiko Sakamoto's Algorithm
      sint16 ye, mo, da, dw;
      static sint8 t[] = {0, 3, 2, 5, 0, 3, 5, 1, 4, 6, 2, 4};
      ye=GPSyear;
      mo=GPSmonth;
      da=GPSday;

       ye -= mo < 3;
      dw=(ye + ye/4 - ye/100 + ye/400 + t[mo-1] + da) % 7;
      switch(dw)
      {

            case 0  :
                  os_sprintf(&wday[0],"Sunday");
                  break;
            case 1  :
                        os_sprintf(&wday[0],"Monday");
                  break;
            case 2  :
                        os_sprintf(&wday[0],"Tuesday");
                  break;
            case 3  :
                        os_sprintf(&wday[0],"Wednesday");
                  break;
            case 4  :
                        os_sprintf(&wday[0],"Thursday");
                  break;
            case 5  :
                        os_sprintf(&wday[0],"Friday");
                  break;
            case 6  :
                        os_sprintf(&wday[0],"Saturday");
                  break;

      }

      // os_sprintf() doesn't like more than one string argument!! ,
os_strcat, strcat throw compiler errors
      // so memcpy is your friend!

      os_memcpy(message, wday,(uint8)os_strlen(&wday[0]));
      os_bzero(message+os_strlen(&wday[0]),(uint8) 1 );
      os_sprintf(&tmpmesg[0],", %s %d, %d %d:%d:%d UTC\n
\r",smonth,GPSday,GPSyear,GPShour,GPSminute,GPSsecond);
      os_memcpy(message+os_strlen(&message[0]), tmpmesg,(uint8)os_strlen
(&tmpmesg[0]));
```

```
      espconn_sent( &socket_tcp_daytime, &message[0], os_strlen(&message[0])
);
      //os_printf("Connection on port 13");
      return;
}

/
******************************************************************************
 * FunctionName : debug_udp_sent_cb
 * Description  : Callback for debug purposes
 * Parameters  : NONE
 * Returns      : NONE

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

void ICACHE_FLASH_ATTR
debug_udp_sent_cb(void *arg)
{
      struct espconn *pespconn = arg;
      os_printf("Debug UDP socket data  sent\n");
}


/
******************************************************************************
 * FunctionName : create_sockets
 * Description  : Create sockets after got wireless connection and IP Address
 * Parameters  : NONE
 * Returns      : NONE

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

void ICACHE_FLASH_ATTR
create_sockets(void)
{
    struct ip_info ipconfig;

   //disarm timer first
    os_timer_disarm(&ipready_timer);

   //get ip info of ESP8266 station
    wifi_get_ip_info(STATION_IF, &ipconfig);

    if (wifi_station_get_connect_status() == STATION_GOT_IP &&
ipconfig.ip.addr != 0)
   {

            //
******************************************************************************
************************
            //          debug socket
            //
******************************************************************************
*************************

            socket_udp_debug.type = ESPCONN_UDP;
            socket_udp_debug.proto.udp = (esp_udp *)os_zalloc(sizeof
(esp_udp));
            socket_udp_debug.proto.udp->local_port = espconn_port();  // set
a available  port
            socket_udp_debug.proto.udp->remote_port = debug_udp_remote_port;
                  os_memcpy(socket_udp_debug.proto.udp->remote_ip,
debug_udp_remote_ip, 4); // ESP8266 udp remote IP

            espconn_regist_sentcb(&socket_udp_debug, debug_udp_sent_cb); //
register a udp packet sent callback
                  espconn_create(&socket_udp_debug);   // create udp

            //
******************************************************************************
************************
            //          daytime service tcp socket    port 13
            //
******************************************************************************
*************************


            socket_tcp_daytime.type = ESPCONN_TCP;
            socket_tcp_daytime.state = ESPCONN_NONE;
            socket_tcp_daytime.proto.tcp = (esp_tcp *)os_zalloc
((uint32)sizeof(esp_tcp));
            socket_tcp_daytime.proto.tcp->local_port = 13;      // daytime
service  port
            espconn_regist_connectcb(&socket_tcp_daytime, tcp_daytime_listen);
            espconn_regist_sentcb(&socket_tcp_daytime, tcp_daytime_sent_cb);
            espconn_accept(&socket_tcp_daytime);
            espconn_regist_time(&socket_tcp_daytime, 30, 0);

            //
******************************************************************************
************************
            //          time service tcp socket       port 37
            //
******************************************************************************
*************************

            socket_tcp_time.type = ESPCONN_TCP;
            socket_tcp_time.state = ESPCONN_NONE;
            socket_tcp_time.proto.tcp = (esp_tcp *)os_zalloc((uint32)sizeof
(esp_tcp));
            socket_tcp_time.proto.tcp->local_port = 37;      // time service
port
            espconn_regist_connectcb(&socket_tcp_time, tcp_time_listen);
            espconn_regist_sentcb(&socket_tcp_time, tcp_time_sent_cb);
            espconn_accept(&socket_tcp_time);
            espconn_regist_time(&socket_tcp_time, 30, 0);

            //
******************************************************************************
************************
            //          ntp service udp socket        port 123
            //
******************************************************************************
*************************

            socket_udp_ntp.type = ESPCONN_UDP;
            socket_udp_ntp.proto.udp = (esp_udp *)os_zalloc(sizeof(esp_udp));
            socket_udp_ntp.proto.udp->local_port = 123;
            espconn_create(&socket_udp_ntp);
            espconn_regist_recvcb(&socket_udp_ntp, udp_ntp_recvcb);

    }
   else
   {
        if ((wifi_station_get_connect_status() == STATION_WRONG_PASSWORD ||
                wifi_station_get_connect_status() == STATION_NO_AP_FOUND ||
                wifi_station_get_connect_status() == STATION_CONNECT_FAIL))
        {
         os_printf("connect fail !!! \r\n");
        }
      else
      {
           //re-arm timer to check ip again
            os_timer_setfn(&ipready_timer, (os_timer_func_t *)create_sockets,
NULL);
            os_timer_arm(&ipready_timer, 100, 0);
        }
    }
}


/
******************************************************************************
 * FunctionName : uart0_baud_setup
 * Description  : Simpler function to setup UART0, enables RX and TX channel
( NO flow control!! )
 * Parameters  : baudrate, databits, parity enabled/disabled, parity type,
stopbits
 *                         If parity is disabled, parity type doesn't matters
 * Returns      : NONE

******************************************************************************
*/
void ICACHE_FLASH_ATTR

uart0_baud_setup(unsigned int baudrate,unsigned char databits,
unsigned char parenabled, unsigned char partype,unsigned char stopbits)
{
    UartDevice    UartConf;
```

```
/* Available baudrate options:
BIT_RATE_300
BIT_RATE_600
BIT_RATE_1200
BIT_RATE_2400
BIT_RATE_4800
BIT_RATE_9600
BIT_RATE_19200
BIT_RATE_38400
BIT_RATE_57600
BIT_RATE_74880
BIT_RATE_115200
BIT_RATE_230400
BIT_RATE_460800
BIT_RATE_921600
BIT_RATE_1843200
BIT_RATE_3686400 */

/* Available databits options
FIVE_BITS
SIX_BITS
SEVEN_BITS
EIGHT_BITS */

/* Available parenabled options
STICK_PARITY_DIS
STICK_PARITY_EN */

/* Available partype options
ODD_BITS
EVEN_BITS */

/* Available stop bits options
ONE_STOP_BIT
ONE_HALF_STOP_BIT
TWO_STOP_BIT */

// TX0 pin enabled
PIN_FUNC_SELECT(PERIPHS_IO_MUX_U0TXD_U, FUNC_U0TXD);
// pullup disabled on TX0 pin
PIN_PULLUP_DIS(PERIPHS_IO_MUX_U0TXD_U);
//U0RXD default behavior RX0

// Pullup disabled on RX0 pin
PIN_PULLUP_DIS(PERIPHS_IO_MUX_U0RXD_U);
// U0RXD default function UART0 RX

// baudrate set
uart_div_modify(0, (uint16)(UART_CLK_FREQ / (uint32)(baudrate) ) );

// Temporal structure to hold values
UartConf.data_bits=databits;
UartConf.exist_parity=parenabled;
```

```
    UartConf.parity=partype;
    UartConf.stop_bits=stopbits;

    // Writing values to UART0 CONF0 register
    WRITE_PERI_REG(UART_CONF0(0), (uint32)UartConf.exist_parity
            | (uint32)UartConf.parity
            | (((uint8)UartConf.stop_bits) << UART_STOP_BIT_NUM_S)
            | (((uint8)UartConf.data_bits) << UART_BIT_NUM_S));

}

/
******************************************************************************
 * FunctionName : connection_data_task
 * Description  : Task for closing tcp comm sockets
 * Parameters   : An event descriptor for the socket to be closed
 *
 * Returns      : NONE

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

static void ICACHE_FLASH_ATTR
connection_data_task(os_event_t *events)
{

      if(events == NULL) {
            return;
      }

      switch(events->sig) {

      case CONN_CLOSE_TCP_DAYTIME:
            espconn_disconnect(&socket_tcp_daytime);
            break;

      case CONN_CLOSE_TCP_TIME:
            espconn_disconnect(&socket_tcp_time);
            break;
      }


}

/
******************************************************************************
 * FunctionName : process_data_task
 * Description  : Task for parsing a valid NMEA $GPRMC sentence that
contains timing information.
 * Parameters   :
 *
 * Returns      : NONE
```

```
******************************************************************************
*/

static void ICACHE_FLASH_ATTR
process_data_task(os_event_t *events)
{
      static uint8 state = 0;
      int32 ret = 0;
      uint32 data_len = 0;
      uint8 sectmp;
      uint8 mintmp;
      uint8 hourtmp;
      uint8 monthtmp;
      uint8 daytmp;
      uint16 msectmp;
      uint16 yeartmp;

      unsigned char GPRMC_time[11] ;
      unsigned char GPRMC_date[7] ;
      unsigned char GPRMC_fix[2] ;

      unsigned short sizefound;
      unsigned short strsize;
      unsigned char count;
      char * gprmc_found;
      char * sep1_found;
      char * sep2_found;

      if(events == NULL) {
            return;
      }

      switch(events->sig) {
      case TRANS_RECV_DATA_FROM_UART:

            // char *os_strstr(const char *haystack, const char *needle);
            gprmc_found =(char *) os_strstr(NMEA_buffer, "$GPRMC");
            if( gprmc_found != NULL  )
            {
                  sizefound=(gprmc_found - NMEA_buffer);
                  // check if found the whole string!! not just a piece
                  if( sizefound+ GPRMC_MAX_SIZE < UART_RX_BUFF_SIZE)
                        {

                              //
$GPRMC,201705.000,A,0000.0000,N,00000.0000,W,1.10,265.50,120816,,,A*79

                              // Parsing time using "," characters #1 and # 2
                              //
**********************************************************
                              sep1_found =(char *) os_strstr(gprmc_found, ","
)+1 ;
```

```
                              sep2_found =(char *) os_strstr(sep1_found, ","
) ;
                              strsize = sep2_found - sep1_found;
                              os_memcpy(GPRMC_time, sep1_found,
(uint8)strsize);
                              os_bzero(GPRMC_time+(char)strsize,(uint8) 1 );
                              //os_printf("GPS Time: %s\r\n",GPRMC_time);


                              sectmp=((uint8)GPRMC_time[4]-(uint8)0x30)*10;
                              sectmp=sectmp+((uint8)GPRMC_time[5]-
(uint8)0x30);
                              //os_printf("GPS Second: %d\r\n",sectmp);

                              mintmp=((uint8)GPRMC_time[2]-(uint8)0x30)*10;
                              mintmp=mintmp+((uint8)GPRMC_time[3]-
(uint8)0x30);
                              //os_printf("GPS Minute: %d\r\n",mintmp);


                              hourtmp=((uint8)GPRMC_time[0]-(uint8)0x30)*10;
                              hourtmp=hourtmp+((uint8)GPRMC_time[1]-
(uint8)0x30);
                              //os_printf("GPS Minute: %d\r\n",hourtmp);

                              // Extract and convert milliseconds ( some GPS
doesn' )
                              if( strsize > 7 )
                              {
                                    msectmp=((uint16)GPRMC_time[7]-
(uint16)0x30)*100;
                                    if(strsize>8)
                                    {
                                          msectmp=msectmp+( (uint16)GPRMC_time
[8]-(uint16)0x30)*10;
                                          if(strsize>9)
                                          {
                                                msectmp=msectmp
+( (uint16)GPRMC_time[9]-(uint16)0x30);
                                          }

                                    }
                              }
                              else
                              {
                                    msectmp=0;
                              }
                              //os_printf("GPS milliSecond: %d\r\n",msectmp);


                              // Parsing date using "," characters #9 and # 10
                              //
**********************************************
```

```
                              sep1_found=gprmc_found;
                              for(count=0 ; count < 9 ; count ++)
                              {
                                    sep2_found =(char *) os_strstr(sep1_found,
"," )+1 ;

                                    sep1_found=sep2_found;
                              }

                              sep2_found =(char *) os_strstr(sep1_found, ","
) ;
                              strsize = sep2_found - sep1_found;
                              os_memcpy(GPRMC_date, sep1_found,
(uint8)strsize);

                              os_bzero(GPRMC_date+(char)strsize,(uint8) 1 );
                              //os_printf("GPS Date: %s\r\n",GPRMC_date);

                              yeartmp=((uint16)GPRMC_date[4]-(uint16)0x30)*10;
                              yeartmp=yeartmp+((uint8)GPRMC_date[5]-
(uint16)0x30);

                              yeartmp=yeartmp+2000;
                              //os_printf("GPS Year: %d\r\n",yeartmp);

                              monthtmp=((uint8)GPRMC_date[2]-(uint8)0x30)*10;
                              monthtmp=monthtmp+((uint8)GPRMC_date[3]-
(uint8)0x30);

                              //os_printf("GPS Month: %d\r\n",monthtmp);

                              daytmp=((uint8)GPRMC_date[0]-(uint8)0x30)*10;
                              daytmp=daytmp+((uint8)GPRMC_date[1]-
(uint8)0x30);

                              //os_printf("GPS Day: %d\r\n",daytmp);

                              // Parsing satellite fix "," characters #2 and
# 3
                              //
**********************************************
                              sep1_found =(char *) os_strstr(gprmc_found, ","
)+1 ;

                              sep2_found = sep1_found;
                              sep1_found =(char *) os_strstr(sep2_found, "," )
+1 ;

                              os_memcpy(GPRMC_fix, sep1_found,(uint8) 1 );
                              os_bzero(GPRMC_fix+(char) 1 ,(uint8) 1 );
                              //os_printf("GPS Fix: %d\r\n",GPSfix);


                              //if seconds changed update global vars and get
uP ticks

                              if(GPSsecond != sectmp)
                              {
                                    // **** update time variables should be
Atomic!!! *****
```

```
                                    // Serial string received, rise watchdog
flag
                                    GPS_wd_flag=1;

                                    if( os_strcmp(GPRMC_fix, "A") == 0 )
                                    {
                                          GPSfix=1;
                                    }
                                    else
                                    {
                                          GPSfix=0;
                                    }

                                    GPSsecond=sectmp;
                                    GPSminute=mintmp;
                                    GPShour=hourtmp;
                                    GPSmillisecond=msectmp;
                                    GPSyear=yeartmp;
                                    GPSmonth=monthtmp;
                                    GPSday=daytmp;
                                    RTCcounts=system_get_time();

                                    // toggle led
                                    if (GPIO_REG_READ(GPIO_OUT_ADDRESS) & (1
<< gpioledpin))
                                    {
                                          gpio_output_set(0, (1 <<
gpioledpin), 0, 0);
                                    }
                                    else
                                    {
                                          gpio_output_set((1 << gpioledpin),
0, 0, 0);
                                    }

                                    //os_printf("RTC counts:%d\r
\n",RTCcounts);
                                    //os_printf("%d:%d:%d %d-%d-%d\r
\n",GPShour,GPSminute,GPSsecond,GPSday,GPSmonth,GPSyear);
                              }

                        }

            }

            // espconn_send(&socket_udp_debug, NMEA_BUFFER[0], os_strlen
(NMEA_BUFFER[0]));
            //os_printf("Buff Data: %s\r\n",NMEA_buffer);
            // os_printf("Str len: %d\r\n",os_strlen(NMEA_BUFFER));

            break;
      default:
            os_printf("sig %d\r\n",events->sig);
            break;
      }
}

/
******************************************************************************
 * FunctionName : user_init
 * Description  : Hardware and other initializations that must be done after
poweron
 * Parameters   :
 *
 * Returns      : NONE

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

void ICACHE_FLASH_ATTR user_init()
{
      // init gpio susbytem
      gpio_init();


      // GPIO PINS SETUP
      // ******************
      // configure GPIO2_U  to be GPIO2, disable pull-up, set as output
      PIN_FUNC_SELECT(PERIPHS_IO_MUX_GPIO2_U, FUNC_GPIO2);
      PIN_PULLUP_DIS(PERIPHS_IO_MUX_GPIO2_U);
      gpio_output_set(0, 0, (1 << gpioledpin), 0);

      // UART0 SETUP
      // *****************
      uart0_baud_setup
(BIT_RATE_4800,EIGHT_BITS,STICK_PARITY_DIS,ODD_BITS,ONE_STOP_BIT);


      //clear all interrupts
      WRITE_PERI_REG(UART_INT_ENA(0), 0x0000);
      // Set interrupt when rx fifo over threshold, and set threshold 35
      uart0_int_setup(UART_RXFIFO_FULL_INT_ENA,40);
      // Set interrupt when rx timeout 10 chars
      uart0_int_setup(UART_RXFIFO_TOUT_INT_ENA,3);
      // Attach int service routine function
      ETS_UART_INTR_ATTACH(uart0_rx_intr_handler,NULL);
      // Enable gobal UART interupts
      ETS_UART_INTR_ENABLE();

      pRxBuffer = Uart_Buf_Init(UART_RX_BUFF_SIZE);

      // WiFi Station setup
      // works as Client connected to a WiFi AP, please modify values in
user_config.h
      // ******************
      char ssid[32] = SSID;
      char password[64] = SSID_PASSWORD;
      char hostname[32] = HOST_NAME;
      struct station_config stationConf;
      //Set station mode
      wifi_set_opmode( 0x1 );
      //Set ap settings
      os_memcpy(&stationConf.ssid, ssid, 32);
      os_memcpy(&stationConf.password, password, 64);
      wifi_station_set_hostname(hostname);
      wifi_station_set_config(&stationConf);

      // Socket setup
      //set a timer to check whether got ip from router succeed or not. If 
suceed create sockets
      os_timer_disarm(&ipready_timer);
      os_timer_setfn(&ipready_timer, (os_timer_func_t *)create_sockets, NULL);
      os_timer_arm(&ipready_timer, 100, 0);

      // create a task that processes UART0 received data
      system_os_task(process_data_task, PROC_TASK_PRIO, trans_Queue, 
TRANS_QUEUE_LEN);

      // create a task that closes socket communications
      system_os_task(connection_data_task, CONN_TASK_PRIO, conn_Queue, 
CONN_QUEUE_LEN);


      // setup GPS watchdog (3000ms, repeating) if no serial string received 
in 3s
      os_timer_setfn(&gps_watchdog, (os_timer_func_t *)gps_wd_func, NULL);
      os_timer_arm(&gps_watchdog, 3000, 1);

}
```

uart.h

```
/
******************************************************************************
 * Copyright 2013-2014 Espressif Systems
 *
 * FileName: uart.h
 *
 * Description: entry file of user application
 *
 * Modification history:
 *     2015/9/24, v1.0 create this file.
******************************************************************************
*/
#ifndef UART_APP_H
#define UART_APP_H

#include "uart_register.h"
#include "eagle_soc.h"
#include "c_types.h"

#define UART0   0
```

```
#define UART1   1

typedef enum {
    FIVE_BITS = 0x0,
    SIX_BITS = 0x1,
    SEVEN_BITS = 0x2,
    EIGHT_BITS = 0x3
} UartBitsNum4Char;

typedef enum {
    ONE_STOP_BIT             = 0x1,
    ONE_HALF_STOP_BIT        = 0x2,
    TWO_STOP_BIT             = 0x3
} UartStopBitsNum;

typedef enum {
    NONE_BITS = 0x2,
    ODD_BITS   = 1,
    EVEN_BITS = 0
} UartParityMode;

typedef enum {
    STICK_PARITY_DIS   = 0,
    STICK_PARITY_EN    = 1
} UartExistParity;

typedef enum {
    UART_None_Inverse = 0x0,
    UART_Rxd_Inverse = UART_RXD_INV,
    UART_CTS_Inverse = UART_CTS_INV,
    UART_Txd_Inverse = UART_TXD_INV,
    UART_RTS_Inverse = UART_RTS_INV,
} UART_LineLevelInverse;


typedef enum {
    BIT_RATE_300 = 300,
    BIT_RATE_600 = 600,
    BIT_RATE_1200 = 1200,
    BIT_RATE_2400 = 2400,
    BIT_RATE_4800 = 4800,
    BIT_RATE_9600   = 9600,
    BIT_RATE_19200  = 19200,
    BIT_RATE_38400  = 38400,
    BIT_RATE_57600  = 57600,
    BIT_RATE_74880  = 74880,
    BIT_RATE_115200 = 115200,
    BIT_RATE_230400 = 230400,
    BIT_RATE_460800 = 460800,
    BIT_RATE_921600 = 921600,
    BIT_RATE_1843200 = 1843200,
    BIT_RATE_3686400 = 3686400,
} UartBautRate;
```

```
typedef enum {
    NONE_CTRL,
    HARDWARE_CTRL,
    XON_XOFF_CTRL
} UartFlowCtrl;

typedef enum {
    USART_HardwareFlowControl_None = 0x0,
    USART_HardwareFlowControl_RTS = 0x1,
    USART_HardwareFlowControl_CTS = 0x2,
    USART_HardwareFlowControl_CTS_RTS = 0x3
} UART_HwFlowCtrl;

typedef enum {
    EMPTY,
    UNDER_WRITE,
    WRITE_OVER
} RcvMsgBuffState;

typedef struct {
    uint32     RcvBuffSize;
    uint8     *pRcvMsgBuff;
    uint8     *pWritePos;
    uint8     *pReadPos;
    uint8      TrigLvl; //JLU: may need to pad
    RcvMsgBuffState  BuffState;
} RcvMsgBuff;

typedef struct {
    uint32   TrxBuffSize;
    uint8   *pTrxBuff;
} TrxMsgBuff;

typedef enum {
    BAUD_RATE_DET,
    WAIT_SYNC_FRM,
    SRCH_MSG_HEAD,
    RCV_MSG_BODY,
    RCV_ESC_CHAR,
} RcvMsgState;

typedef struct {
    UartBautRate       baut_rate;
    UartBitsNum4Char  data_bits;
    UartExistParity      exist_parity;
    UartParityMode          parity;    // chip size in byte
    UartStopBitsNum   stop_bits;
    UartFlowCtrl         flow_ctrl;
    RcvMsgBuff          rcv_buff;
    TrxMsgBuff           trx_buff;
    RcvMsgState        rcv_state;
    int                      received;
    int                      buff_uart_no;  //indicate which uart use tx/rx
buffer
} UartDevice;

void uart_init(UartBautRate uart0_br);
///////////////////////////////////////
#define UART_FIFO_LEN  128  //define the tx fifo length
//
#define UART_TX_EMPTY_THRESH_VAL 0x10
#define UART_TX_BUFFER_SIZE 8192
#define URAT_TX_LOWER_SIZE 1024*7
#define URAT_TX_UPPER_SIZE  8000*1

#define UART_RX_BUFFER_SIZE  (10*1024) //201

//#define UART_TX_DEBUG
#define DBG(...)
#define DBG1(...) //uart1_sendStr_no_wait
#define DBG2(...) //os_printf

struct UartBuffer {
    uint16     UartBuffSize;
    uint8     *pUartBuff;
    uint8     *pInPos;
    uint8     *pOutPos;
    STATUS  BuffState;
    uint16    Space;  //remanent space of the buffer
//    uint8  TcpControl;
    struct  UartBuffer       *nextBuff;
};

struct UartRxBuff {
    uint32     UartRxBuffSize;
    uint8     *pUartRxBuff;
    uint8     *pWritePos;
    uint8     *pReadPos;
    STATUS RxBuffState;
    uint32    Space;  //remanent space of the buffer
} ;

void tx_buff_enq(const char *pdata, uint16 data_len, bool force);
uint16 rx_buff_deq(char *pdata, uint16 data_len);
//==============================================
#define FUNC_UART0_CTS 4
#define UART_LINE_INV_MASK  (0x3f<<19)

#endif
```

uart_register.h

```
/
*****************************************************************************
 * Copyright 2013-2014 Espressif Systems
 *
 * FileName: uart_register.h
 *
 * Description: entry file of user application
 *
 * Modification history:
 *     2015/9/24, v1.0 create this file.
******************************************************************************
*/

#ifndef UART_REGISTER_H_
#define UART_REGISTER_H_

#define REG_UART_BASE(i)                (0x60000000ul + (i)*0xf00)
//version value:32'h062000

#define UART_FIFO(i)                    (REG_UART_BASE(i) + 0x0)
#define UART_RXFIFO_RD_BYTE                 0x000000FF
#define UART_RXFIFO_RD_BYTE_S               0

#define UART_INT_RAW(i)                 (REG_UART_BASE(i) + 0x4)
#define UART_RXFIFO_TOUT_INT_RAW            (BIT(8))
#define UART_BRK_DET_INT_RAW                (BIT(7))
#define UART_CTS_CHG_INT_RAW                (BIT(6))
#define UART_DSR_CHG_INT_RAW                (BIT(5))
#define UART_RXFIFO_OVF_INT_RAW             (BIT(4))
#define UART_FRM_ERR_INT_RAW                (BIT(3))
#define UART_PARITY_ERR_INT_RAW             (BIT(2))
#define UART_TXFIFO_EMPTY_INT_RAW           (BIT(1))
#define UART_RXFIFO_FULL_INT_RAW            (BIT(0))

#define UART_INT_ST(i)                  (REG_UART_BASE(i) + 0x8)
#define UART_RXFIFO_TOUT_INT_ST             (BIT(8))
#define UART_BRK_DET_INT_ST                 (BIT(7))
#define UART_CTS_CHG_INT_ST                 (BIT(6))
#define UART_DSR_CHG_INT_ST                 (BIT(5))
#define UART_RXFIFO_OVF_INT_ST              (BIT(4))
#define UART_FRM_ERR_INT_ST                 (BIT(3))
#define UART_PARITY_ERR_INT_ST              (BIT(2))
#define UART_TXFIFO_EMPTY_INT_ST            (BIT(1))
#define UART_RXFIFO_FULL_INT_ST             (BIT(0))

#define UART_INT_ENA(i)                 (REG_UART_BASE(i) + 0xC)
#define UART_RXFIFO_TOUT_INT_ENA            (BIT(8))
#define UART_BRK_DET_INT_ENA                (BIT(7))
#define UART_CTS_CHG_INT_ENA                (BIT(6))
#define UART_DSR_CHG_INT_ENA                (BIT(5))
#define UART_RXFIFO_OVF_INT_ENA             (BIT(4))
#define UART_FRM_ERR_INT_ENA                (BIT(3))
#define UART_PARITY_ERR_INT_ENA             (BIT(2))
#define UART_TXFIFO_EMPTY_INT_ENA           (BIT(1))
#define UART_RXFIFO_FULL_INT_ENA            (BIT(0))

#define UART_INT_CLR(i)                 (REG_UART_BASE(i) + 0x10)
#define UART_RXFIFO_TOUT_INT_CLR            (BIT(8))
```

```
#define UART_BRK_DET_INT_CLR                (BIT(7))
#define UART_CTS_CHG_INT_CLR                (BIT(6))
#define UART_DSR_CHG_INT_CLR                (BIT(5))
#define UART_RXFIFO_OVF_INT_CLR             (BIT(4))
#define UART_FRM_ERR_INT_CLR                (BIT(3))
#define UART_PARITY_ERR_INT_CLR             (BIT(2))
#define UART_TXFIFO_EMPTY_INT_CLR           (BIT(1))
#define UART_RXFIFO_FULL_INT_CLR            (BIT(0))

#define UART_CLKDIV(i)                  (REG_UART_BASE(i) + 0x14)
#define UART_CLKDIV_CNT                     0x000FFFFF
#define UART_CLKDIV_S                       0

#define UART_AUTOBAUD(i)                (REG_UART_BASE(i) + 0x18)
#define UART_GLITCH_FILT                    0x000000FF
#define UART_GLITCH_FILT_S                  8
#define UART_AUTOBAUD_EN                    (BIT(0))

#define UART_STATUS(i)                  (REG_UART_BASE(i) + 0x1C)
#define UART_TXD                            (BIT(31))
#define UART_RTSN                           (BIT(30))
#define UART_DTRN                           (BIT(29))
#define UART_TXFIFO_CNT                     0x000000FF
#define UART_TXFIFO_CNT_S                   16
#define UART_RXD                            (BIT(15))
#define UART_CTSN                           (BIT(14))
#define UART_DSRN                           (BIT(13))
#define UART_RXFIFO_CNT                     0x000000FF
#define UART_RXFIFO_CNT_S                   0

#define UART_CONF0(i)                   (REG_UART_BASE(i) + 0x20)
#define UART_DTR_INV                        (BIT(24))
#define UART_RTS_INV                        (BIT(23))
#define UART_TXD_INV                        (BIT(22))
#define UART_DSR_INV                        (BIT(21))
#define UART_CTS_INV                        (BIT(20))
#define UART_RXD_INV                        (BIT(19))
#define UART_TXFIFO_RST                     (BIT(18))
#define UART_RXFIFO_RST                     (BIT(17))
#define UART_IRDA_EN                        (BIT(16))
#define UART_TX_FLOW_EN                     (BIT(15))
#define UART_LOOPBACK                       (BIT(14))
#define UART_IRDA_RX_INV                    (BIT(13))
#define UART_IRDA_TX_INV                    (BIT(12))
#define UART_IRDA_WCTL                      (BIT(11))
#define UART_IRDA_TX_EN                     (BIT(10))
#define UART_IRDA_DPLX                      (BIT(9))
#define UART_TXD_BRK                        (BIT(8))
#define UART_SW_DTR                         (BIT(7))
#define UART_SW_RTS                         (BIT(6))
#define UART_STOP_BIT_NUM                   0x00000003
#define UART_STOP_BIT_NUM_S                 4
#define UART_BIT_NUM                        0x00000003
#define UART_BIT_NUM_S                      2
```

```
#define UART_PARITY_EN                      (BIT(1))
#define UART_PARITY                         (BIT(0))

#define UART_CONF1(i)                   (REG_UART_BASE(i) + 0x24)
#define UART_RX_TOUT_EN                     (BIT(31))
#define UART_RX_TOUT_THRHD                  0x0000007F
#define UART_RX_TOUT_THRHD_S                24
#define UART_RX_FLOW_EN                     (BIT(23))
#define UART_RX_FLOW_THRHD                  0x0000007F
#define UART_RX_FLOW_THRHD_S                16
#define UART_TXFIFO_EMPTY_THRHD             0x0000007F
#define UART_TXFIFO_EMPTY_THRHD_S           8
#define UART_RXFIFO_FULL_THRHD              0x0000007F
#define UART_RXFIFO_FULL_THRHD_S            0

#define UART_LOWPULSE(i)                (REG_UART_BASE(i) + 0x28)
#define UART_LOWPULSE_MIN_CNT               0x000FFFFF
#define UART_LOWPULSE_MIN_CNT_S             0

#define UART_HIGHPULSE(i)               (REG_UART_BASE(i) + 0x2C)
#define UART_HIGHPULSE_MIN_CNT              0x000FFFFF
#define UART_HIGHPULSE_MIN_CNT_S            0

#define UART_PULSE_NUM(i)               (REG_UART_BASE(i) + 0x30)
#define UART_PULSE_NUM_CNT                  0x0003FF
#define UART_PULSE_NUM_CNT_S                0

#define UART_DATE(i)                    (REG_UART_BASE(i) + 0x78)
#define UART_ID(i)                      (REG_UART_BASE(i) + 0x7C)

#endif // UART_REGISTER_H_INCLUDED
```

uart.c

```
/
******************************************************************************
 * Copyright 2013-2014 Espressif Systems
 *
 * FileName: uart.c
 *
 * Description: entry file of user application
 *
 * Modification history:
 *     2015/9/24, v1.0 create this file.
******************************************************************************
*/
#include "ets_sys.h"
#include "osapi.h"
#include "driver/uart.h"
#include "osapi.h"
#include "driver/uart_register.h"
#include "mem.h"
#include "user_interface.h"
```

```
#include "espconn.h"

#define FUNC_U1TXD_BK                   2
#define FUNC_U0CTS                             4
extern UartDevice    UartDev;


static struct UartBuffer *pTxBuffer = NULL;
static struct UartBuffer *pRxBuffer = NULL;

LOCAL void uart0_rx_intr_handler(void *para);

extern struct espconn *get_trans_conn(void);
void uart1_sendStr_no_wait(const char *str);
struct UartBuffer * Uart_Buf_Init(uint32 buf_size);
void set_tcp_block(void);
void clr_tcp_block(void);
void rx_buff_enq(void);
void tx_start_uart_buffer(uint8 uart_no);
void uart_rx_intr_disable(uint8 uart_no);
void uart_rx_intr_enable(uint8 uart_no);
/
******************************************************************************
 * FunctionName : uart_config
 * Description  : Internal used function
 *                UART0 used for data TX/RX, RX buffer size is 0x100,
interrupt enabled
 *                UART1 just used for debug output
 * Parameters   : uart_no, use UART0 or UART1 defined ahead
 * Returns      : NONE

******************************************************************************
*/
LOCAL void ICACHE_FLASH_ATTR
uart_config(uint8 uart_no)
{
    if (uart_no == UART1) {
        PIN_FUNC_SELECT(PERIPHS_IO_MUX_GPIO2_U, FUNC_U1TXD_BK);
    } else {
        /* rcv_buff size if 0x100 */
        ETS_UART_INTR_ATTACH(uart0_rx_intr_handler,  &(UartDev.rcv_buff));
        PIN_PULLUP_DIS(PERIPHS_IO_MUX_U0TXD_U);
        PIN_FUNC_SELECT(PERIPHS_IO_MUX_U0TXD_U, FUNC_U0TXD);
        PIN_FUNC_SELECT(PERIPHS_IO_MUX_MTDO_U, FUNC_U0RTS);
        PIN_FUNC_SELECT(PERIPHS_IO_MUX_MTCK_U, FUNC_U0CTS);
        PIN_PULLUP_DIS(PERIPHS_IO_MUX_MTCK_U);
    }

    uart_div_modify(uart_no, (uint16)(UART_CLK_FREQ / (uint32)
(UartDev.baut_rate)));

    WRITE_PERI_REG(UART_CONF0(uart_no), (uint32)UartDev.exist_parity
            | (uint32)UartDev.parity
            | (((uint8)UartDev.stop_bits) << UART_STOP_BIT_NUM_S)
            | (((uint8)UartDev.data_bits) << UART_BIT_NUM_S));

    //clear rx and tx fifo,not ready
    SET_PERI_REG_MASK(UART_CONF0(uart_no), UART_RXFIFO_RST | UART_TXFIFO_RST);
    CLEAR_PERI_REG_MASK(UART_CONF0(uart_no), UART_RXFIFO_RST |
UART_TXFIFO_RST);

    if (uart_no == UART0) {
        //set rx fifo trigger
        WRITE_PERI_REG(UART_CONF1(uart_no),
                ((100 & UART_RXFIFO_FULL_THRHD) << UART_RXFIFO_FULL_THRHD_S) |
                ((110 & UART_RX_FLOW_THRHD) << UART_RX_FLOW_THRHD_S) |
                UART_RX_FLOW_EN |
                (0x02 & UART_RX_TOUT_THRHD) << UART_RX_TOUT_THRHD_S |
                UART_RX_TOUT_EN |

                ((0x10 & UART_TXFIFO_EMPTY_THRHD) <<
UART_TXFIFO_EMPTY_THRHD_S)); //wjl
                //SET_PERI_REG_MASK( UART_CONF0(uart_no),UART_TX_FLOW_EN);  //
add this sentense to add a tx flow control via MTCK( CTS )

        SET_PERI_REG_MASK(UART_INT_ENA(uart_no), UART_RXFIFO_TOUT_INT_ENA |
                UART_FRM_ERR_INT_ENA);
    } else {
        WRITE_PERI_REG(UART_CONF1(uart_no),
                ((UartDev.rcv_buff.TrigLvl & UART_RXFIFO_FULL_THRHD) <<
UART_RXFIFO_FULL_THRHD_S));
    }

    //clear all interrupt
    WRITE_PERI_REG(UART_INT_CLR(uart_no), 0xffff);
    //enable rx_interrupt
    SET_PERI_REG_MASK(UART_INT_ENA(uart_no), UART_RXFIFO_FULL_INT_ENA |
UART_RXFIFO_OVF_INT_ENA);
}

/
******************************************************************************
 * FunctionName : uart1_tx_one_char
 * Description  : Internal used function
 *                Use uart1 interface to transfer one char
 * Parameters   : uint8 TxChar - character to tx
 * Returns      : OK

******************************************************************************
*/
STATUS
uart_tx_one_char(uint8 uart, uint8 TxChar)
{
    for(;;) {
        uint32 fifo_cnt = READ_PERI_REG(UART_STATUS(uart)) &
(UART_TXFIFO_CNT << UART_TXFIFO_CNT_S);

        if ((fifo_cnt >> UART_TXFIFO_CNT_S & UART_TXFIFO_CNT) < 126) {
            break;
        }
    }

    WRITE_PERI_REG(UART_FIFO(uart) , TxChar);
    return OK;
}

/
******************************************************************************
 * FunctionName : uart1_write_char
 * Description  : Internal used function
 *                Do some special deal while tx char is '\r' or '\n'
 * Parameters   : char c - character to tx
 * Returns      : NONE

******************************************************************************
*/
LOCAL void ICACHE_FLASH_ATTR
uart1_write_char(char c)
{
    if (c == '\n') {
        (void)uart_tx_one_char(UART1, '\r');
        (void)uart_tx_one_char(UART1, '\n');
    } else if (c == '\r') {
    } else {
        (void)uart_tx_one_char(UART1, (uint8)c);
    }
}

/
******************************************************************************
 * FunctionName : uart0_rx_intr_handler
 * Description  : Internal used function
 *                UART0 interrupt handler, add self handle code inside
 * Parameters   : void *para - point to ETS_UART_INTR_ATTACH's arg
 * Returns      : NONE

******************************************************************************
*/
LOCAL void
uart0_rx_intr_handler(void *para)
{
    uint8 uart_no = UART0;
    if (UART_FRM_ERR_INT_ST == (READ_PERI_REG(UART_INT_ST(uart_no)) &
UART_FRM_ERR_INT_ST)) {
        uart1_sendStr_no_wait("FRM_ERR\r\n");
        WRITE_PERI_REG(UART_INT_CLR(uart_no), UART_FRM_ERR_INT_CLR);
    }

    if (UART_RXFIFO_FULL_INT_ST == (READ_PERI_REG(UART_INT_ST(uart_no)) &
UART_RXFIFO_FULL_INT_ST)) {
        uart_rx_intr_disable(uart_no);
        rx_buff_enq();
    } else if (UART_RXFIFO_TOUT_INT_ST == (READ_PERI_REG(UART_INT_ST(uart_no))
& UART_RXFIFO_TOUT_INT_ST)) {
        uart_rx_intr_disable(uart_no);
        rx_buff_enq();
    } else if (UART_TXFIFO_EMPTY_INT_ST == (READ_PERI_REG(UART_INT_ST
(uart_no)) & UART_TXFIFO_EMPTY_INT_ST)) {
        CLEAR_PERI_REG_MASK(UART_INT_ENA(uart_no), UART_TXFIFO_EMPTY_INT_ENA);
        tx_start_uart_buffer(uart_no);
        WRITE_PERI_REG(UART_INT_CLR(uart_no), UART_TXFIFO_EMPTY_INT_CLR);
   } else if (UART_RXFIFO_OVF_INT_ST == (READ_PERI_REG(UART_INT_ST(uart_no)) &
UART_RXFIFO_OVF_INT_ST)) {
        uart1_sendStr_no_wait("RX OVF!\r\n");
        WRITE_PERI_REG(UART_INT_CLR(uart_no), UART_RXFIFO_OVF_INT_CLR);
    }
}

/
******************************************************************************
 * FunctionName : uart_init
 * Description  : user interface for init uart
 * Parameters   : UartBautRate uart0_br - uart0 bautrate
 *                UartBautRate uart1_br - uart1 bautrate
 * Returns      : NONE

******************************************************************************
*/
void ICACHE_FLASH_ATTR
uart_init(UartBautRate uart0_br)
{
    UartDev.baut_rate = uart0_br;
    uart_config(UART0);
    UartDev.baut_rate = BIT_RATE_115200;
    uart_config(UART1);
    ETS_UART_INTR_ENABLE();

    os_install_putc1(uart1_write_char);

    pTxBuffer = Uart_Buf_Init(UART_TX_BUFFER_SIZE);
    pRxBuffer = Uart_Buf_Init(UART_RX_BUFFER_SIZE);
}

/
******************************************************************************
 * FunctionName : uart_tx_one_char_no_wait
 * Description  : uart tx a single char without waiting for fifo
 * Parameters   : uint8 uart - uart port
 *                uint8 TxChar - char to tx
 * Returns      : STATUS

******************************************************************************
*/
STATUS
uart_tx_one_char_no_wait(uint8 uart, uint8 TxChar)
{
    uint8 fifo_cnt = ((READ_PERI_REG(UART_STATUS(uart)) >> UART_TXFIFO_CNT_S)&
UART_TXFIFO_CNT);

    if (fifo_cnt < 126) {
        WRITE_PERI_REG(UART_FIFO(uart) , TxChar);
    }

    return OK;
}

/
******************************************************************************
 * FunctionName : uart1_sendStr_no_wait
 * Description  : uart tx a string without waiting for every char, used for
print debug info which can be lost
 * Parameters   : const char *str - string to be sent
 * Returns      : NONE

******************************************************************************
*/
void
uart1_sendStr_no_wait(const char *str)
{
    if(str == NULL) {
        return;
    }

    while (*str) {
        (void)uart_tx_one_char_no_wait(UART1, (uint8)*str);
        str++;
    }
}
/
******************************************************************************
 * FunctionName : Uart_Buf_Init
 * Description  : tx buffer enqueue: fill a first linked buffer
 * Parameters   : char *pdata - data point  to be enqueue
 * Returns      : NONE

******************************************************************************
*/
struct UartBuffer *ICACHE_FLASH_ATTR
Uart_Buf_Init(uint32 buf_size)
{
    uint32 heap_size = system_get_free_heap_size();

    if(buf_size > 65535) { // now not support
        DBG1("no buf for uart\n\r");
        return NULL;
    }
    if (heap_size <= buf_size) {
        DBG1("no buf for uart\n\r");
        return NULL;
    } else {
        DBG("test heap size: %d\n\r", heap_size);
        struct UartBuffer *pBuff = (struct UartBuffer *)os_malloc
((uint32)sizeof(struct UartBuffer));
        pBuff->UartBuffSize = (uint16)buf_size; // THIS OK
        pBuff->pUartBuff = (uint8 *)os_malloc(pBuff->UartBuffSize);
        pBuff->pInPos = pBuff->pUartBuff;
        pBuff->pOutPos = pBuff->pUartBuff;
        pBuff->Space = pBuff->UartBuffSize;
        pBuff->BuffState = OK;
        pBuff->nextBuff = NULL;
        //        pBuff->TcpControl = RUN;
        return pBuff;
    }
}

LOCAL void
Uart_Buf_Cpy(struct UartBuffer *pCur, const char *pdata , uint16 data_len)
{
    if ((pCur == NULL) || (pdata == NULL) || (data_len == 0)) {
        return ;
    }

    uint16 tail_len = (uint16)(pCur->pUartBuff + pCur->UartBuffSize - pCur-
>pInPos); // THIS OK

    if (tail_len >= data_len) { //do not need to loop back  the queue
        os_memcpy(pCur->pInPos , pdata , data_len);
        pCur->pInPos += (data_len);
        pCur->pInPos = (pCur->pUartBuff + (pCur->pInPos - pCur->pUartBuff)
% pCur->UartBuffSize);
        pCur->Space -= data_len;
    } else {
        os_memcpy(pCur->pInPos, pdata, tail_len);
        pCur->pInPos += (tail_len);
        pCur->pInPos = (pCur->pUartBuff + (pCur->pInPos - pCur->pUartBuff)
% pCur->UartBuffSize);
        pCur->Space -= tail_len;
        os_memcpy(pCur->pInPos, pdata + tail_len , data_len - tail_len);
        pCur->pInPos += (data_len - tail_len);
        pCur->pInPos = (pCur->pUartBuff + (pCur->pInPos - pCur->pUartBuff)
% pCur->UartBuffSize);
        pCur->Space -= (data_len - tail_len);
    }

}

/
******************************************************************************
 * FunctionName : uart_buf_free
 * Description  : deinit of the tx buffer
 * Parameters   : struct UartBuffer* pTxBuff - tx buffer struct pointer
 * Returns      : NONE
```

```
******************************************************************************
*/
void ICACHE_FLASH_ATTR
uart_buf_free(struct UartBuffer *pBuff)
{
    if(pBuff != NULL) {
        if(pBuff->pUartBuff != NULL) {
            os_free(pBuff->pUartBuff);
        }
        os_free(pBuff);
    }
}

uint16 ICACHE_FLASH_ATTR
rx_buff_get_data_len(void)
{
    uint16 buf_len = pRxBuffer->UartBuffSize - pRxBuffer->Space;
    return buf_len;
}

uint16 ICACHE_FLASH_ATTR
rx_buff_deq(char *pdata, uint16 data_len)
{
    uint16 len_tmp = 0;

    uint16 buf_len = pRxBuffer->UartBuffSize - pRxBuffer->Space;
    uint16 tail_len = (uint16)(pRxBuffer->pUartBuff + pRxBuffer-
>UartBuffSize - pRxBuffer->pOutPos); // THIS OK
    len_tmp = ((data_len > buf_len) ? buf_len : data_len);

    if (pRxBuffer->pOutPos <= pRxBuffer->pInPos) {
        if(pdata != NULL) {
            os_memcpy(pdata, pRxBuffer->pOutPos, len_tmp);
        }
        pRxBuffer->pOutPos += len_tmp;
        pRxBuffer->Space += len_tmp;
    } else {
        if (len_tmp > tail_len) {
            if(pdata != NULL) {
                os_memcpy(pdata, pRxBuffer->pOutPos, tail_len);
            }
            pRxBuffer->pOutPos += tail_len;
            pRxBuffer->pOutPos = (pRxBuffer->pUartBuff + (pRxBuffer->pOutPos
- pRxBuffer->pUartBuff) % pRxBuffer->UartBuffSize);
            pRxBuffer->Space += tail_len;

            if(pdata != NULL) {
                os_memcpy(pdata + tail_len , pRxBuffer->pOutPos, len_tmp -
tail_len);
            }
            pRxBuffer->pOutPos += (len_tmp - tail_len);
            pRxBuffer->pOutPos = (pRxBuffer->pUartBuff + (pRxBuffer->pOutPos
- pRxBuffer->pUartBuff) % pRxBuffer->UartBuffSize);
```

```
            pRxBuffer->Space += (len_tmp - tail_len);
        } else {
            if(pdata != NULL) {
                os_memcpy(pdata, pRxBuffer->pOutPos, len_tmp);
            }
            pRxBuffer->pOutPos += len_tmp;
            pRxBuffer->pOutPos = (pRxBuffer->pUartBuff + (pRxBuffer->pOutPos
- pRxBuffer->pUartBuff) % pRxBuffer->UartBuffSize);
            pRxBuffer->Space += len_tmp;
        }
    }

    // os_printf("recv:%d\r\n",pRxBuffer->Space);
    return len_tmp;
}

void
rx_buff_enq(void)
{
    uint8 fifo_len = 0, buf_idx = 0,loop;
    ETSParam par = 0;
    uint8 fifo_data;
    uint8* tail = (pRxBuffer->pUartBuff + pRxBuffer->UartBuffSize);
    fifo_len = (READ_PERI_REG(UART_STATUS(UART0)) >>
UART_RXFIFO_CNT_S)&UART_RXFIFO_CNT;

    if(fifo_len > pRxBuffer->Space) {
        buf_idx = pRxBuffer->Space;
    } else {
        buf_idx = fifo_len;
    }

    loop = buf_idx;
    while (loop--) {
        *(pRxBuffer->pInPos++) = READ_PERI_REG(UART_FIFO(UART0)) & 0xFF;
        if (pRxBuffer->pInPos == tail) {
            pRxBuffer->pInPos = pRxBuffer->pUartBuff;
        }
    }

    fifo_len -= buf_idx;

    while (fifo_len--) {
        fifo_data = READ_PERI_REG(UART_FIFO(UART0)) & 0xFF; // discard data
    }

    pRxBuffer->Space -= buf_idx ;
    par = buf_idx;
    if(system_os_post(TRANS_TASK_PROI, (ETSSignal)TRANS_RECV_DATA_FROM_UART,
par) != TRUE) {
        os_printf("post fail!!!\n\r");
    }
    WRITE_PERI_REG(UART_INT_CLR(UART0), UART_RXFIFO_FULL_INT_CLR |
UART_RXFIFO_TOUT_INT_CLR);
```

```
    uart_rx_intr_enable(UART0);
}

void ICACHE_FLASH_ATTR
tx_buff_enq(const char *pdata, uint16 data_len, bool force)
{
    CLEAR_PERI_REG_MASK(UART_INT_ENA(UART0), UART_TXFIFO_EMPTY_INT_ENA);
    //DBG2("len:%d\n\r",data_len);
    if(pdata != NULL) {
        if (pTxBuffer == NULL) {
            DBG1("\n\rnull, create buffer struct\n\r");
            pTxBuffer = Uart_Buf_Init(UART_TX_BUFFER_SIZE);

            if (pTxBuffer != NULL) {
                Uart_Buf_Cpy(pTxBuffer ,  pdata,  data_len);
            } else {
                DBG1("uart tx MALLOC no buf \n\r");
            }
        } else {
            if (data_len <= pTxBuffer->Space) {
                Uart_Buf_Cpy(pTxBuffer ,  pdata,  data_len);
            } else if (force) {
                for(;;) {
                    tx_start_uart_buffer(UART0);
                    CLEAR_PERI_REG_MASK(UART_INT_ENA(UART0),
UART_TXFIFO_EMPTY_INT_ENA);
                    if (data_len <= pTxBuffer->Space) {
                        Uart_Buf_Cpy(pTxBuffer ,  pdata,  data_len);
                        break;
                    }
                    ets_delay_us(70);
                    WRITE_PERI_REG(0X60000914, 0x73);
                };
            } else {
                DBG1("UART TX BUF FULL!!!!\n\r");
            }
        }

        if ((pTxBuffer != NULL) && (pTxBuffer->Space <= URAT_TX_LOWER_SIZE)) {
            set_tcp_block();
        }
    }

    SET_PERI_REG_MASK(UART_CONF1(UART0), (UART_TX_EMPTY_THRESH_VAL &
UART_TXFIFO_EMPTY_THRHD) << UART_TXFIFO_EMPTY_THRHD_S);
    SET_PERI_REG_MASK(UART_INT_ENA(UART0), UART_TXFIFO_EMPTY_INT_ENA);
}

//--------------------------------
LOCAL void tx_fifo_insert(struct UartBuffer *pTxBuff, uint8 data_len,  uint8
uart_no)
{
    uint8 i;
    if(pTxBuff == NULL) {
        return;
    }

    for (i = 0; i < data_len; i++) {
        WRITE_PERI_REG(UART_FIFO(uart_no) , *(pTxBuff->pOutPos++));

        if (pTxBuff->pOutPos == (pTxBuff->pUartBuff + pTxBuff->UartBuffSize))
{
            pTxBuff->pOutPos = pTxBuff->pUartBuff;
        }
    }

    pTxBuff->pOutPos = (pTxBuff->pUartBuff + (pTxBuff->pOutPos - pTxBuff-
>pUartBuff) % pTxBuff->UartBuffSize);
    pTxBuff->Space += data_len;
}

/
******************************************************************************
 * FunctionName : tx_start_uart_buffer
 * Description  : get data from the tx buffer and fill the uart tx fifo, co-
work with the uart fifo empty interrupt
 * Parameters   : uint8 uart_no - uart port num
 * Returns      : NONE

******************************************************************************
*/
void tx_start_uart_buffer(uint8 uart_no)
{
    uint8 tx_fifo_len = (READ_PERI_REG(UART_STATUS(uart_no)) >>
UART_TXFIFO_CNT_S)&UART_TXFIFO_CNT;
    uint8 fifo_remain = UART_FIFO_LEN - tx_fifo_len ;
    uint8 len_tmp;
    uint32 data_len;

    if (pTxBuffer) {
        data_len = (pTxBuffer->UartBuffSize - pTxBuffer->Space);

        if (data_len > fifo_remain) {
            len_tmp = fifo_remain;
            tx_fifo_insert(pTxBuffer, len_tmp, uart_no);
            SET_PERI_REG_MASK(UART_INT_ENA(UART0), UART_TXFIFO_EMPTY_INT_ENA);
        } else {
            len_tmp = (uint8)data_len; // THIS OK
            tx_fifo_insert(pTxBuffer, len_tmp, uart_no);

        }

        if (pTxBuffer->Space >= URAT_TX_UPPER_SIZE) {
            (void)system_os_post(TRANS_TASK_PROI,
(ETSSignal)TRANS_SEND_DATA_TO_UART_OVER,(ETSParam)0);
        }
    }
```

```
}

void uart_rx_intr_disable(uint8 uart_no)
{
    CLEAR_PERI_REG_MASK(UART_INT_ENA(uart_no), UART_RXFIFO_FULL_INT_ENA |
UART_RXFIFO_TOUT_INT_ENA);
}

void uart_rx_intr_enable(uint8 uart_no)
{
    SET_PERI_REG_MASK(UART_INT_ENA(uart_no), UART_RXFIFO_FULL_INT_ENA |
UART_RXFIFO_TOUT_INT_ENA);
}


void ICACHE_FLASH_ATTR
set_tcp_block(void)
{
    struct espconn * trans_conn = (struct espconn *)get_trans_conn();
    if(trans_conn == NULL) {

    } else if (trans_conn->type == ESPCONN_TCP) {
        DBG1("TCP BLOCK\n\r");
        (void)espconn_recv_hold(trans_conn);
        DBG2("b space: %d\n\r", pTxBuffer->Space);
    } else {

    }
}

void ICACHE_FLASH_ATTR
clr_tcp_block(void)
{
    struct espconn * trans_conn = (struct espconn *)get_trans_conn();
    if(trans_conn == NULL) {

    } else if (trans_conn->type == ESPCONN_TCP) {
        DBG1("TCP recover\n\r");
        (void)espconn_recv_unhold(trans_conn);
        DBG2("r space: %d\n\r", pTxBuffer->Space);
    } else {

    }

}
//========================================================
```