# Application Note: Interfacing with Wifi module and Arduino over UART

The Arduino or WiFi module make an ideal platform for prototyping and data collection with the GDK101.

## Wiring with external WiFi module

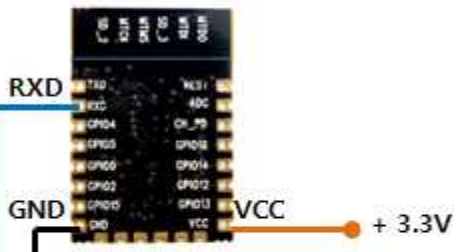
The first step is to wire up the GDK101 to WiFi module. This example shows the simple example about transmission from the GDK101.

* Make the following Connections:

| Wire | GDK101 | ESP8266MOD |
|---|---|---|
| Red (5V) | pin #3, 5V | - |
| Black (GND) | pin #4, GND | GND |
| Orange (3.3V) | - | VCC |
| Blue | pin #1, TX | RXD |

**- GDK101 -**


**- WiFi module :** ESP8266MOD **-**




* When GDK101 is turned on, the GDK101 send the Measured value(1min) to the host at every 1 min automatically.

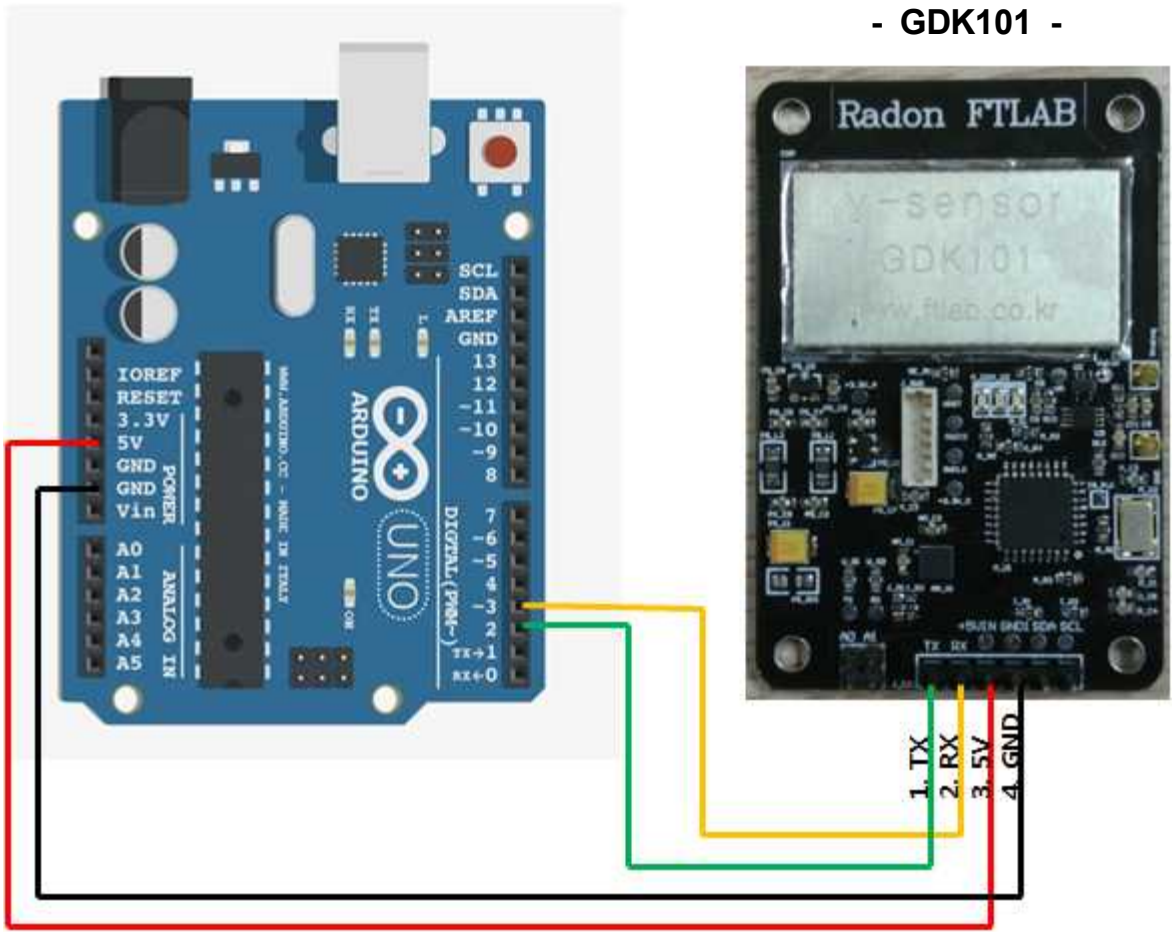
* Sample packet : ¬ M:1.24<CR><LF> (see page 3)

## Wiring with Arduino UNO

The first step is to wire up the GDK101 to your Arduino. This example shows the Arduino UNO with a GDK101.

### * Make the following Connections:

| Wire | Arduino | GDK101 |
|---|---|---|
| Red | 5V | pin #3, 5V |
| Black | GND | pin #4, GND |
| Yellow | Digital pin #3, TX | pin #2, UART RX |
| Green | Digital pin #2, RX | pin #1, UART TX |



# Command packet (ASCII)

## Mode

| Mode | Description |
|---|---|
| Auto_send mode | When GDK101 is turned on, the GDK101 send the Measured value(1min) to the host at every 1 min automatically. |
| Normal mode | If other request is sent from the host(except for 'R'), the GDK101 will be changed to Normal mode.(Should request comes in response.) |

## Packet Format

| STX | Command | : | Data | CR | LF |
|---|---|---|---|---|---|

| Field | Length | Type | Description |
|---|---|---|---|
| STX | 1 byte | Character | " ┐ " |
| Command | 1 byte | Character | Select a command from the Command List. |
| : | 1 byte | Character | This symbol separate the Command field and the Data field |
| Data | variable | Character string | '?' (when requesting to GDK101)<br>Data bytes(when you receive a response from GDK101) |
| CR, LF | 2 byte | Binary data | Used to identify the end of a packet |

* Sample packet
  - Request : ┐ D:?<CR><LF>
  - Response : ┐ D:1.24<CR><LF>

## Command List

| Command | Meaning | Description |
|---|---|---|
| D | Measured value 10min | Return gamma value(10min avg, 1min update) |
| M | Measured value 1min | Return gamma value(1min avg, 1min update) |
| T | Measuring time | Return the current measuring time |
| S | Status | Return the current status |
| F | Firmware version | Return firmware version of GDK101 |
| V | Vibration status | Return the current vibration status |
| R | Reset | Reset GDK101 |
| U | Auto send enable | Set auto send status(enable / disable) |
| A | All data | Return all data of GDK101(S+T+D+E) |

Example

| Packet | | Meaning | |
|---|---|---|---|
| Request<br>Response | ┐ D:?<CR><LF><br>┐ D:1.24<CR><LF> | Measured value (10min) is 1.24 uSv/h | Range<br>0.00 ~ 99.99 uSv/h |
| *Request<br>Response | ┐ M:?<CR><LF> or automatically<br>┐ M:1.24<CR><LF> | Measured value (1min) is 1.24 uSv/h | Range<br>0.00 ~ 99.99 uSv/h |
| Request<br>Response | ┐ T:?<CR><LF><br>┐ T:124 13:44:53<CR><LF> | Measuring time is '124day 13hour 44min 53sec' | Type<br>Day HH:mm:ss |
| Request<br>Response | ┐ S:?<CR><LF><br>┐ S:1<CR><LF> | Status is '1' | Status<br>0: Power on ~ 10sec<br>1: 10sec to 10min<br>2: after 10min |
| Request<br>Response | ┐ F:?<CR><LF><br>┐ F:v0.6<CR><LF> | GDK101 firmware version is 'v0.6' | - |
| Request<br>Response | ┐ V:?<CR><LF><br>┐ V:1<CR><LF> | The current vibration status is 1 | Status<br>0: No vibration<br>1: Detect vibration |
| Request<br>Response | ┐ R:1<CR><LF><br>┐ R:1<CR><LF> | Reset success | - |
| Request<br>Response | ┐ U:1<CR><LF><br>no response | Auto send is 'enable' | Status<br>1: Enable<br>Other request send (except for 'R') : Disable |
| Request<br>Response | ┐ A:?<CR><LF><br>┐ A:1/3 22:15:46/0.55/1.67<CR><LF> | 1. Status is '1'<br>2. Measuring time is '3day 22hour 15min 46sec'<br>3. Measured value(10min) is 0.55 uSv/h<br>4. Measured value(1min) is 1.67 uSv/h | - |

* If status of Auto_send is 'enable', the GDK101 send the Measured value(1min) to the host at every 1 min automatically. Default status of Auto_send is 'enable'.

## Writing the Code

The next step is to write the code to drive the device. We will show a quick and simple sketch that will report the gamma value, and then we will show a sketch that does the same job.

We will use SoftwareSerial library, Timer library and mthread library to interface with the GDK101.

Import it into the project and initialize it in the setup() routine:

```
#include <SoftwareSerial.h>
#include <Timer.h>        // https://github.com/JChristensen/Timer
#include <mthread.h>      // https://github.com/jlamothe/mthread

/* Thread class for receive UART */
class FooThread : public Thread
{
   public: FooThread(int id);
   protected: bool loop();
   private: int id;
};

FooThread::FooThread(int id){
   this->id = id;
}

/*  Pin map  ::  Digital 2 - RX (to Gamma Sensor TX),  Digital 3 - TX (to Gamma Sensor RX)  */
SoftwareSerial mySerial(2, 3);
Timer ts1;
char rec_data[50];                    // Array for received command
bool request_flag = true;             // enable or disable request automatically

void setup() {
   Serial.begin(9600);                // PC - Arduino
   mySerial.begin(9600);              // Arduino - Gamma Sensor
   ts1.every(1000, RequestData);      // Request to Gamma Sensor

   Gamma_INIT();
}
```

We will start the operation of the GDK101 using the threads.

```
bool FooThread::loop()
{
   if(id == 1) RecUartData();         // Thread 1 - for Receive Data
   else ts1.update();                 // Thread 2 - for Timers

   return true;
}
```

# Appendix A: Sample Code

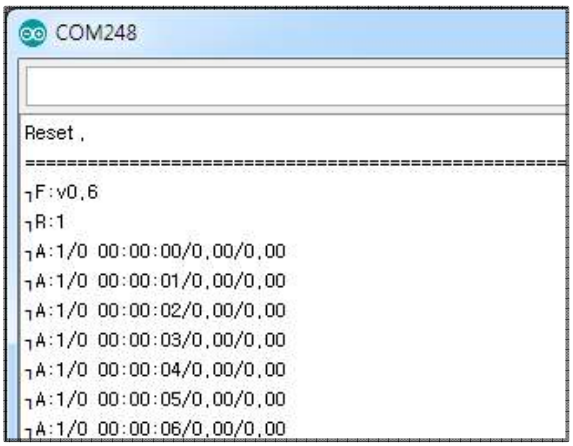
```
#include <SoftwareSerial.h>
#include <Timer.h>        // https://github.com/JChristensen/Timer
#include <mthread.h>     // https://github.com/jlamothe/mthread

/* Thread class for receive UART */
class FooThread : public Thread
{
   public: FooThread(int id);
   protected: bool loop();
   private: int id;
};

FooThread::FooThread(int id){
   this->id = id;
}

/* Command */
enum {
   cmd_GAMMA_RESULT_QUERY          = 0x44,   // D, Read measuring value(10min avg, 1min update)
   cmd_GAMMA_RESULT_QUERY_1MIN     = 0x4D,   // M, Read measuring value(1min avg, 1min update)
   cmd_GAMMA_PROC_TIME_QUERY       = 0x54,   // T, Read measuring time
   cmd_GAMMA_MEAS_QUERY            = 0x53,   // S, Read status
   cmd_GAMMA_FW_VERSION_QUERY      = 0x46,   // F, Read firmware version
   cmd_GAMMA_VIB_STATUS_QUERY      = 0x56,   // V, Response vibration status
   cmd_GAMMA_RESET                 = 0x52,   // R, Reset
   cmd_GAMMA_AUTO_SEND             = 0x55,    // U, Set Auto_send status
   cmd_GAMMA_ALL_QUERY             = 0x41,   // A, Read all data
};

/*  Pin map  :  Digital 2 - RX (to Gamma Sensor TX), Digital 3 - TX (to Gamma Sensor RX)  */
SoftwareSerial mySerial(2, 3);
Timer ts1;
char rec_data[50];                  // Array for received command
bool request_flag = true;           // enable or disable send command automatically

void setup() {
   Serial.begin(9600);              // PC - Arduino
   mySerial.begin(9600);            // Arduino - Gamma Sensor
   ts1.every(1000, RequestData);    // Request to Gamma Sensor
   Gamma_INIT();
}

bool FooThread::loop(){
   if(id == 1) RecUartData();       // Thread 1 - for Receive Data
   else ts1.update();               // Thread 2 - for Timers
   return true;
}
```

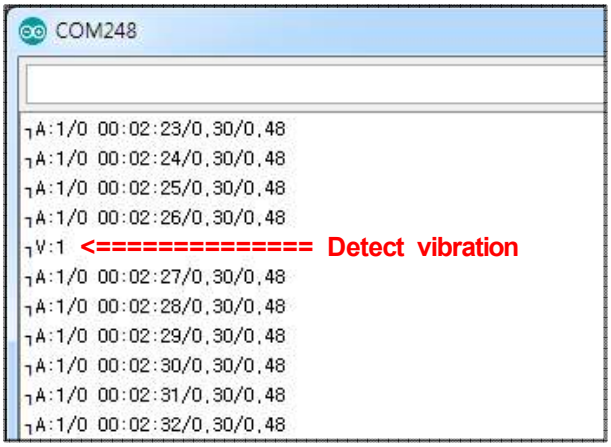
```
/* Gamma Sensor Initialize  */
void Gamma_INIT() {
  main_thread_list->add_thread(new FooThread(1));    // Add Thread 1 for UART Response
  Read_FW();                  // Read FW version
  Reset();                    // Reset
  if(request_flag) main_thread_list->add_thread(new FooThread(2));      // Add Thread 2 for Timers
  Serial.println("=====================================================");
}

/* Meawurement Reset */
void Reset(){
  byte send_data[6] = {0x02, cmd_GAMMA_RESET, ':', '1', 0x0D, 0x0A};
  mySerial.write(send_data, 6);
  Serial.println("Reset.");
  delay(100);
}

/* Read Firmware */
void Read_FW(){
  byte send_data[6] = {0x02, cmd_GAMMA_FW_VERSION_QUERY, ':', '?', 0x0D, 0x0A};
  mySerial.write(send_data, 6);
  delay(100);
}

/* Read all data (automatically) */
void RequestData(){
  byte send_data[6] = {0x02, cmd_GAMMA_ALL_QUERY, 0x3A, 0x3F, 0x0D, 0x0A};
  mySerial.write(send_data, 6);
}

void RecUartData(){
  int rec_size = mySerial.available();
  if (rec_size > 0) {
    for (int i = 0; i < rec_size; i++)
    {
      rec_data[i] = mySerial.read();
      Serial.print(rec_data[i]);
    }
  }
}
```

# Result

- Auto_send Mode

```
bool request_flag = false;                    // disable request automatically
```




- Normal Mode

```
bool request_flag = true;                     // enable request automatically
```