Fejlesztés és kódgenerálás ESP8266 mikrovezérlőre Yakindu segítségével

A dokumentum célja, hogy röviden bemutassa, hogyan lehet a Yakindu eszköz segítségével állapotdiagramokból kódot generálni ESP8266 alapú eszközökre.

1. Bevezetés

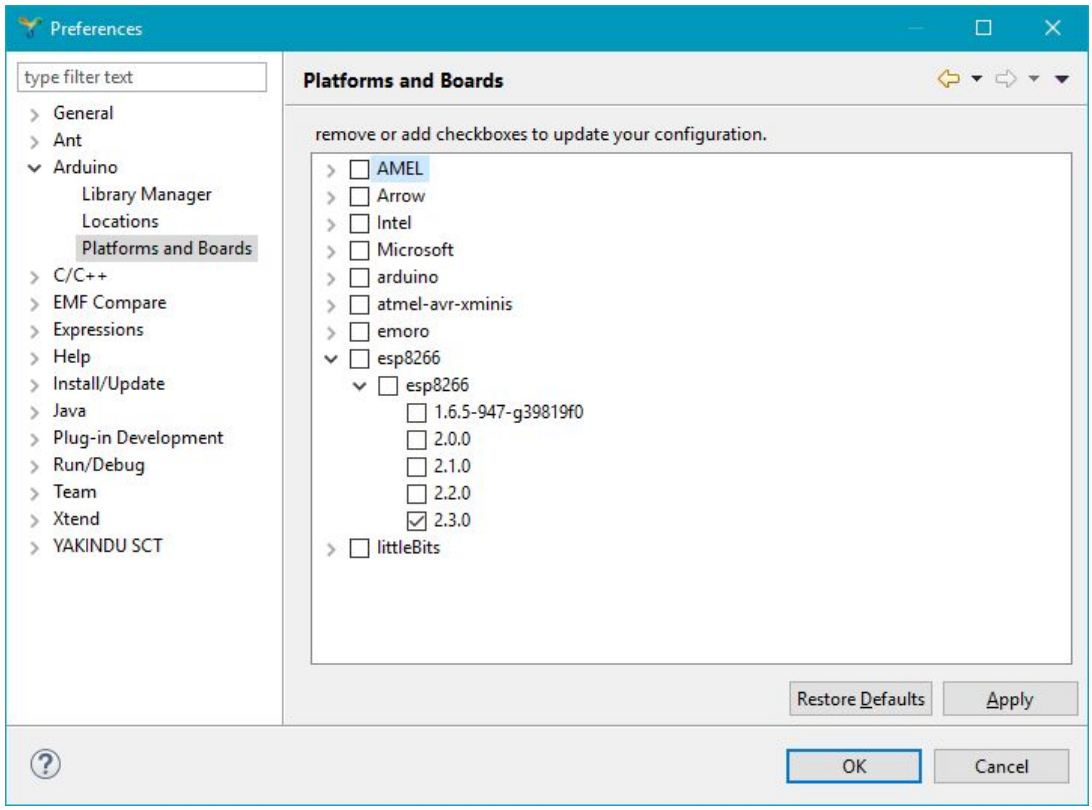
Az ESP8266 egy kínai gyártó az Espressif Systems által gyártott WiFi képes mikrovezérlő aminek a nagy előnye az hogy nagyon olcsó, viszont hátránya, hogy aluldokumentált. Ezt orvosolandó több harmadik féltől származó nyílt forrású SDK jelent meg ehhez a mikrovezérlőhöz. Többek között, létrehozták a hozzá való arduino core-t aminek a segítségével kvázi arduino eszközként lehet fejleszteni az ESP8266 mikrovezérlő alapú eszközökre is. A vonatkozó hardverleírók, core fájlok, letölthetőek a GitHub-ról.

A Yakindu egy Eclipse alapú eszköz ami többek között alkalmas arra hogy állapotgépekből C kódot generáljon. Mivel Eclipse alapú, így rendelkezésre állnak az Eclipse-hez fejlesztett bővítmények. Egy ilyen bővítmény a Jan Baeyens által fejlesztett Arduino plugin. Ezt a plugint fogjuk használni a Yakindu SCT-vel.

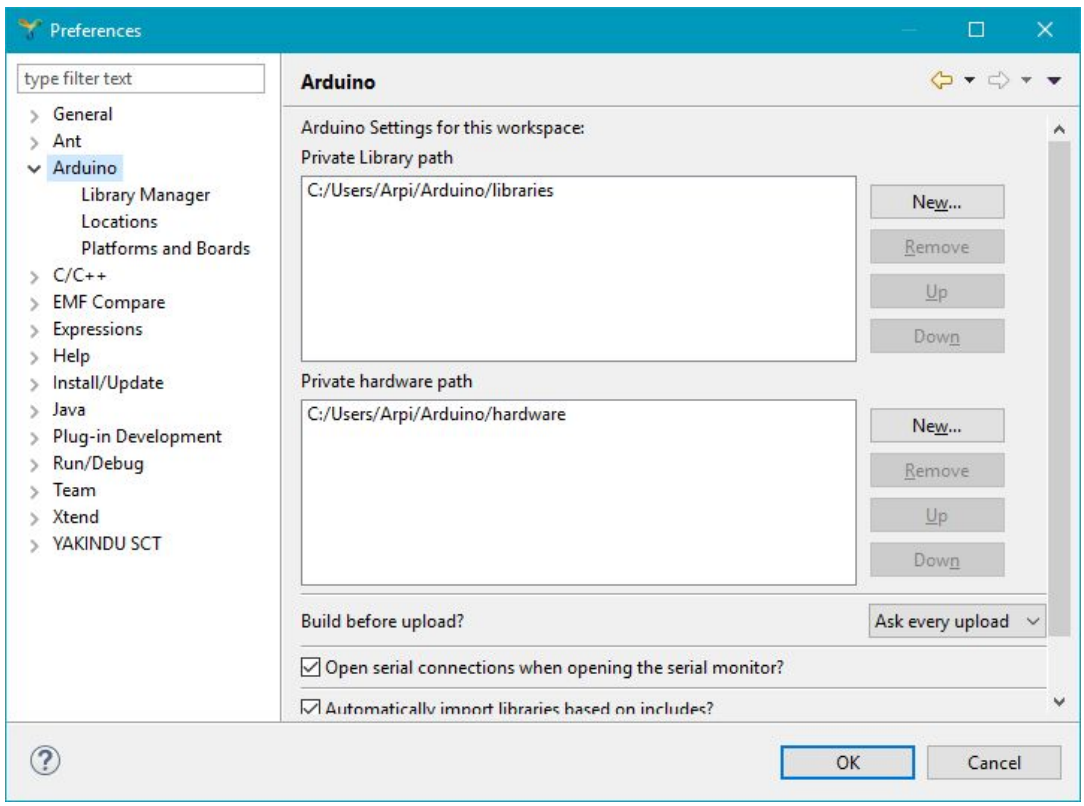
2. Előkészületek, ESP8266 modulok beállítása mint arduino

A következő lépésekre lesz szükség, hogy a Yakindu SCT-ből közvetlenül tudjuk programozni az ESP modulunkat:

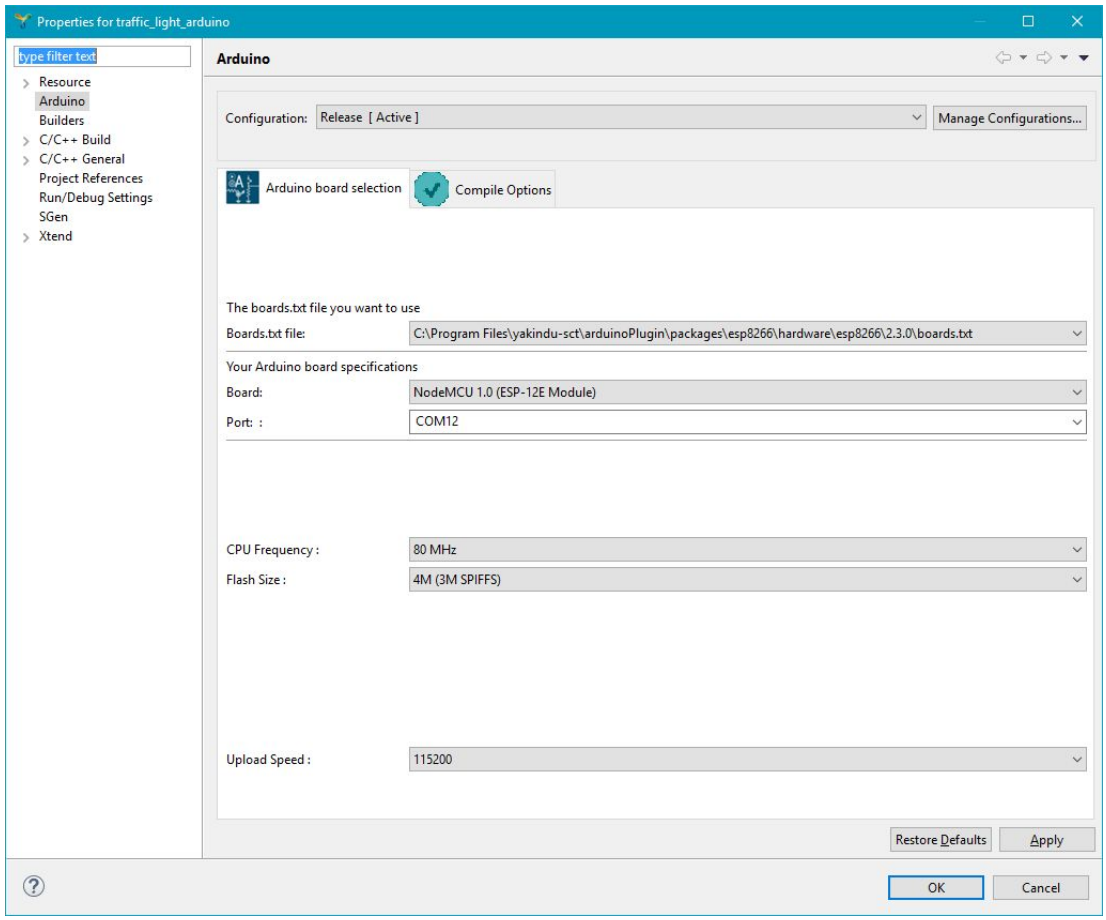
1. Töltsük le a legfrissebb Arduino IDE-t az Arduino hivatalos oldaláról. Én ebben a pillanatban az 1.6.9-es verziót használom, de legalább 1.6.4-es verzió vagy annál újabb szükkséges.
2. A Board Manager segítségével telepítsük az esp8266-hoz tartozó fájlokat a GitHub oldalukon leírtak alapján.
3. Töltsük le a legfrissebb Yakindu-t a hivatalos oldalukról.
4. Telepítsük a fentebb említett Arduino plugint. Ennek a menetét itt nem részletezném, ezt már René Beckmann megtette előttem ebben a leírásban, de egyenlőre még Arduino boardot ne adjunk hozzá a projekthez.
5. A Yakindun belül keressük meg a *Window->Preferences->Arduino->Platform and Boards* menüpontot, és ott jelöljük be az esp8266 opción belül a legújabbat.



Ha ez nem sikerül akkor nem találta meg a plugin automatikusan az Arduino IDE Board Managere által letöltött hardvareleíró fájlokat. Ilyenkor ezeket manuálisan a Yakindun belül a *Window->Preferences->Arduino* menüpontban új *Private hardware path* hozzáadásával tudjuk megtenni.



6. Ha ez sikerült, hozzzáadhatjuk a projekthez a modult. Ezt a projekten a jobb egérgombbal kattintva a *Properties* menüponton belül tehetjük meg: a *Boards.txt file* legördülő menüben válaszuk ki az ESP8266-hoz tartozó Boards.txt-t majd állítsuk be az eszközünkhöz a megfelelő konfigurációt, és a soros portot amelyikre az eszköz csatlakoztatva van.

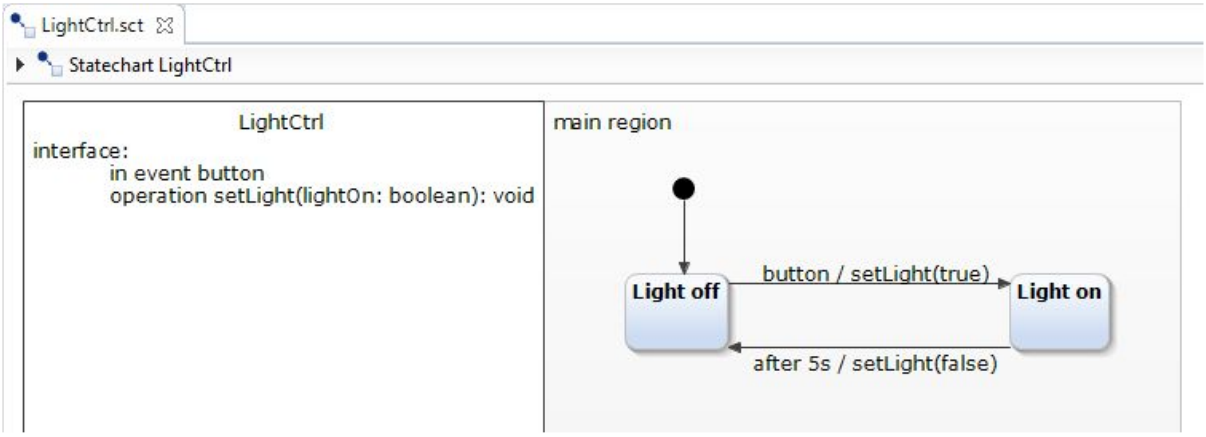


Ha ezeket a lépéseket megtettük, sikeresen tudunk kódot feltölteni az általunk választott ESP modulra. Az én személyes választásom egy Ai-Thinker által gyártott ESP-12E allapú NodeMCU 1.0 -val teljesen megegyező klónra esett, a leírás további részeit ezen az eszközön teszteltem. Sajnos ennél az eszköznél egy kis pluszmunkára van szükség, mert a GPIO pinek számozása nem egyezik meg a lapra nyomtatott számozással.

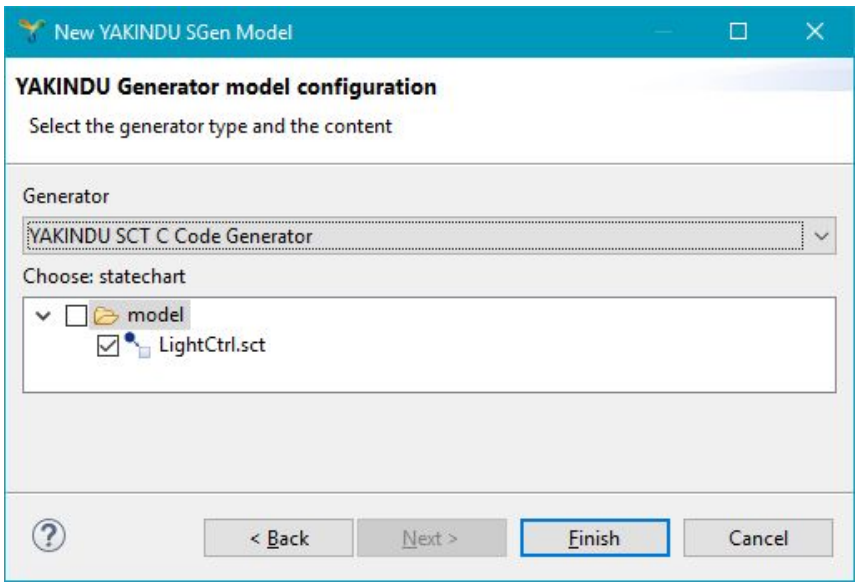
3. ESP8266-tal kompatibilis C kód generálása

A beágyazott rendszerek “HelloWord!”-je a “Blink”. Kezdjünk mi is ennek egy kicsit bonyolultabb változatával, egy lépcsőházvilágítás-vezérlő programmal. Az eredeti projekt szintén az első fejezetben említett René Beckmann-tól származik, arduinohoz készült, az instructables oladalán találjátok meg. Röviden, a lámpa alapvetően le van kapcsolva, ha megnyomjuk a villanykapcsolót felkapcsol, majd bizonyos idő elteltével lekapcsol.

Csináljunk egy új Arduino projectet, abba egy *model* mappát majd ebbe a mappába hozzunk létre egy *Statechart Model*-t LightCtrl néven. A projektünkhöz tartozó állapotgép két állaptból áll egy *Light on* állapotból, amikor a led ég, és egy *Light off* állapotból amikor a led nem ég. Két lehetséges eseményünk van, a gomb lenyomása, és az időzítő lejárta, valamint egy műveletünk, a lámpa kapcsolása. Vegyük föl ezeket a bal oldalon az interfészekhez, az időzítőt nem kell, azt alapból felismeri a rendszer. Ezek után felvehetjük őket a modellünkre is, *esemény / művelet* formában. *Light off* esetben gombnyomásra ugrunk a *Light on* állapotba és végrehajtjuk a felvett lámpa kapcsolás műveletet *true* paraméterrel. *Light on* esetben az időzítő lejártakor ugrunk a *Light off* állapotba lámpa kapcsolás műveletet *false* paraméterrel. Az időzítőt *after *s* formában tudjuk megadni, ahol a * a másodpercek száma. Én 5 másodprcre állítottam az időzítőt, hogy a teszteléskor ne kelljen sokat várni.



Következő lépésként a *model* mappába megcsináljuk a kódgenerátort, ezzel legenerálva a kódot. Ezt a *model*-re jobb egérgombbal kattintva a *New->Code Generator Model* menüpont alatt tehetjük meg. Ne felejtsük el a *Generator* legördülő menüben a generátort C nyelvűre állítani, valamint a megfelelő (jelen esetben az egyetlen) diagram mellett a jelölőnégyzetet kipipálni. Ha változtatunk a modellen, akkor a generátorra jobb egérgombbal kattintva a *Generate Code Artifacts* menüvel tudjuk újrageneráltatni a kódot.



4. A generált kód ESP-hez igazítása

A korábbi lépéseket végrehajtva, meg is kaptuk az állapotgépünket megvalósító C nyelven írt forráskódot. Itt jönnek a problémák, ugyanis a késleltetés számolásához timereket azaz időzítőket kell használnunk. Mivel az időzítők használata szinte mindig platformfüggő a Yakindu nem tudja generálni ezeket, tehát ezt nekünk kell biztosítani. Gondolhatnánk hogy sebaj, létezik library arduinohoz ami ami a rajta lévő hardveres időzítők segítségével megvalósítja a nekünk kellő funkciókat. Ez így igaz is lenne, de az arduinok nagyrésze az Atmel ATmega328-as mikrovezérlőre épül, a mi eszközünk pedig ESP8266 alapú így az alacsonyszintű hardevers funkciók elérése például a hardveres időzítő használata teljesen máshogy működik, így az arduinora készült könyvtárak nem kompatibilisek a mi eszközünkkel. Mivel az ESP8266-hoz megfelelő könyvtár még nem létezett, írtam egyet, ezt fogjuk használni ebben a projektben. ESPTimerService néven a fájlok és a leírás megtalálhatóak a githubon.

Most pár lépésben nézzük meg, miket kell megvalósítanunk az arduino sketch-ben ahhoz hogy az állapotgépünk működjön. Az alapvető arduinos dolgokra mint a pinek kezelése, sketch felépítése stb. nem térek ki, ezekhez rengeteg leírás található az interneten.

Vizsgáljuk meg a generált fájlokat. A *LightCtrl.c* és a *LightCtrl.h* gyakorlatilag az állapotgépet leíró kódot tartalmazzák a *LightCtrlRequired.h* pedig azoknak a függvényeknek a definícióját amiket nekünk kell implementálni ahhoz hogy a program megfelelően működjön.

Első körben létrehozzuk modell (*LightCtrl*) valamint az *ESPTimerService* egy példányát globális változóként majd a setup() függvényben inicializáljuk őket.

```
static LightCtrl lightctrl;
static ESPTimerService timerService;

void setup()
{
	Serial.begin(115200);
	pinMode(BUTTON_PIN, INPUT);
	pinMode(LED_PIN, OUTPUT);
	ESPTS_init(&timerService);
	lightCtrl_init(&lightctrl);   //initialize statechart
	lightCtrl_enter(&lightctrl);  //enter the statechart
	delay(0);
}
```

Ezek után valósítsuk meg a *LightCtrlRequired.h* -ban leírt függvényeket. Itt van az interfész függvénye amit a modellben beállítottunk, ez az a függvény ami le illetve fel kapcsolja a lámpát. Ennek a megvalósítása a legegyszerűbb, a bool paraméter alapján a

LED_PIN lábat vagy logikai magasra, vagy logikai alacsonyra állítjuk, ezzel kapcsolva a rákötött lámpát.

```
void lightCtrlIface_setLight(const LightCtrl* handle, const sc_boolean lightOn) {
		if(lightOn){
			digitalWrite(LED_PIN, HIGH);
		}else{
			digitalWrite(LED_PIN, LOW);
		}
	}
```

A másik két függvény a timerek kezelésére szolgál, a setTimer a paramétereknek megfelelő timer elindítására, az unsetTimer pedig a második paraméterben adott azonosítójú timer megállítására. A setTimer-ben létrehozunk egy timert a megfelelő paraméterek továbbadásával (időtartam, id, ismétlés, callback függvény pointere), majd elindítjuk azt. Amikor az időzítő tüzel, meghívja a paraméterben kapott *void (timerCallback)(void*)* szignatúrájú callback függvényt, úgy, hogy a *void** paraméter a timer azonosítója integer formátumban, de *void**-á kasztolva. Ha belenézünk a *LightCtrl.h*-ba megtaláljuk a *void lightCtrl_raiseTimeEvent(const LightCtrl* handle, sc_eventid evid);* függvényt. Ez az a függvény amit meg kell hívnunk amikor az időzítő tüzel, de mivel nem egyezik a paraméterlistája azzal amit a timer vár, saját callback függvényt kell írnunk a megfelelő szignatúrával, ezt kell adni az időzítőnek, majd ebből a függvényből kell meghívni az eseményt az állapotgép számára jelző lightCtrl_raiseTimeEvent függvényt. Ez a függvény jelen esetben a *void ESPTimerCallback(void* pArg);* ne felejtsük el, hogy a void* típusú argumentum egy integer számot takar, a callback-et hívó timer azonosítóját, így ezt a további munka ellőtt integerré kell kasztolni.

```
void ESPTimerCallback(void* pArg){
	int id = (int)pArg;
	lightCtrl_raiseTimeEvent(&lightctrl,(sc_eventid)id);
}

void lightCtrl_setTimer(	LightCtrl* handle, const sc_eventid evid,
				const sc_integer time_ms, const sc_boolean periodic){

	ESPTS_createTimer(	&timerService, (int)time_ms, (int)evid,
				periodic, (timerCallback)&ESPTimerCallback);
	ESPTS_setTimer(&timerService, (int)evid);
}
```

Az unsetTimer megvalósítása ennél jóval egyszerűbb, elég a kapott azonosítóval meghívni a *timerService ESPTS_unsetTimer* függvényét.

```
void lightCtrl_unsetTimer(LightCtrl* handle, const sc_eventid evid) {
	ESPTS_unsetTimer(&timerService, (int)evid);
}
```

Ahogyan az időzítő lejártakor bekövetkező timer eseményt nekünk kellett jelezni úgy a gomb lenyomásakor bekövetkező button eseményt is nekünk kell jeleznünk. Mint az előbb, az erre való függvényt is a *LightCtrl.h* -ban találjuk meg, a következő néven: *void lightCtrlIface_raise_button(LightCtrl* handle);*. Nincs más dolgunk mint monitorozni azt a pint amelyikre a gombot kötöttük és ha logikai magas értéket olvasunk ki, akkor meghívjuk a fennt említett függvényt.

```
void loop()
{
	unsigned long current_millies = millis();

	if(digitalRead(BUTTON_PIN) == HIGH){
		lightCtrlIface_raise_button(&lightctrl);
	}

	if ( last_cycle_time == 0L || (current_millies >= last_cycle_time + CYCLE_PERIOD)){
		lightCtrl_runCycle(&lightctrl);
		last_cycle_time = current_millies;
		}
	delay(0);
}
```

Ami kimaradt még, az állapotgép létetése a *void lightCtrl_runCycle(*LightCtrl);* függvénnyel, én ezt minden 10ms-ban egyszer teszem meg.

Az utolsó sor, a *delay(0);* nagyon fontos hogy ott legyen, ugyanis a 0 paraméter ellenére átadja a processzoridjét, hogy más is tudja használni azt, ne csak az arduino keretrendszer. Ha ezt lefelejtjük, akkor a WDT (watchdog timer) azt fogja hinni hogy végtelen ciklusba került a program és újraindítja az eszközt. Mivel ez más kellően hosszú cpu foglalás esetén is bekövetkezhet, így a hosszabb máveletekbe érdemes betűzdelni néhány *delay(0);* sort.

Ezzel el is készült a kód, feltölthetjük az eszközre, ha a megfelelő lábra egy gombot csatlakoztatunk, valamint a másik lábra egy ledet, akkor megfigyelhetjük hogy ahogy azt a modellben leírtuk, a gomb megnyomásakor a led kigyullad, majd öt másodperc után elalszik.

5. IoT, használjunk WiFi-t

Az ESP8266-nak az ára mellett az a legnagyobb előnye, hogy WiFi képes. Ha már van rá lehetőségünk, hogy a projektünket az internetre kössük, tegyük is ezt meg. Az előző projektet úgy fogjuk kiegészíteni a továbbiakban, hogy ne gombnyomásra kapcsoljon fel a lámpa, hanem egy bizonyos http-szerű üzenet hatására.

6. Lehetséges problémák

1. Külső arduinohoz készült library használatakor "avr/pgmspace.h: No such file or directory" hibaüzenet.
   Megoldás: A library cpp fáljaiban ahol a hibá(ka)t dobja, a következő sort:
   ```
   #include <avr\pgmspace.h>
   ```
   át kell írni erre:
   ```
   #if (defined(__AVR__))
   #include <avr\pgmspace.h>
   #else
   #include <pgmspace.h>
   #endif
   ```
   [Forrás]
2. A NodeMCU (és valószinűleg bármelyik ESP8266-os modul) GPIO lábai lebegnek, nincsenek se a földre, se tápra húzva, ezt gombok használatakor figyelembe kell venni és a megfelelő fel/le-húzó ellenállást el kell helyezni az áramkörben.
3. A GPIO lábak számozása nem feltétlenül van szinkronban a lapra nyomtatott lábszámozással, eltérés esetén mindig a GPIO számozás a mérvadó. Pl.: az általam használt NodeMCU lapon a DigitalWrite(4,HIGH); sort leírva azon a lábon mérhetünk logikai magas szintet amelyik a nyákon D2-vel van jelölve, nem pedig azon amelyik D4-el.
4. Ha a hosszabb műveletekben vagy a loop() függvényben nincsen delay(0) hívás, a watchdog timer végtelen ciklusnak hiszi és újraindítja az eszközt.

Hibákat, megjegyzéseket az arpifejes92@gmail.com e-mail címre lehet küldeni.
Fejes Árpád
K0PY1V
2016.11.29