Лілка

Andrew Dunai

квіт. 30, 2024

Зміст:

Попередження: Ця документація знаходиться в розробці. Інформація є неповною, може бути недостовірною і/або застарілою, і може значно змінюватися в майбутньому.

Розділ 1

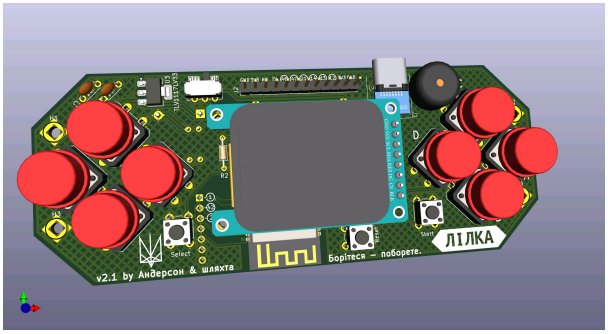
Про проєкт

1.1 Що таке Лілка?

Лілка - це консоль на базі мікроконтролера ESP32-CS-WROOM-1-N16R8.

Її головна особливість - це те, що її можна легко зібрати з готових модулів, які продаються в магазинах.

Інший важливий фактор - це ціна: сумарна вартість всіх компонентів базового набору Лілки в Україні - близько 500-600 гривень.



Мета цього пристрою - це насамперед навчання. Зібравши її, в вас в руках опиниться проста, але повноцінна залізяка з купою цікавих можливостей:

- Емулятор NES, який дозволить вам грати в нінтендівські ігри
- Можливість запускати DOOM з непоганим FPS
- Вбудований WiFi- та Bluetooth-передавач
- Роз'єм розширення для під'єднання власних модулів
- Підтримка Lua для легкого створення власних ігор та програм
- Можливість запуску програм з SD-карти

Попередження: Слід зауважити, що Лілка не позиціонується як ігрова консоль.

Вона - це DIY-набір, який буде дуже легко зібрати навіть недосвідченим початківцям, і її мета - дати вам дешевий набір заліза, з яким можна погратись, а також готову бібліотеку взаємодії з дисплеєм, SD-картою, кнопками, звуком, батареєю та іншими компонентами.

Звісно ж, ви зможете грати на ній в ігри! Але ігри - це не основне її призначення.

Розділ 2

Технічні дані

2.1 Мікроконтролер

Лілка використовує мікроконтролер ESP32-S3-WROOM-1-N16R8. На відміну від ESP32-S3, «WROOM» - це модуль, який містить в собі не лише сам мікроконтролер ESP32-S3, а й флеш-пам'ять, додаткову оперативну пам'ять, кристал, антену та інші компоненти, необхідні для повноцінної роботи мікроконтролера. Це дозволяє значно спростити розробку плати, оскільки вся «обв'язка» мікроконтролера вже виконана на модулі, а також полегшити збір, оскільки модуль можна встановити на плату за допомогою простого SMD-монтажу звичайним паяльником.

ESP32-S3 ESP32-S3-WROOM-1

Характеристики ESP32-S3 наступні:

- Частота процесора: 240 МГц
- Кількість ядер: 2
- Оперативна пам'ять: 512 КБ
- Вбудований Wi-Fi (802.11 b/g/n)
- Вбудований Bluetooth (BLE)
- 45 GPIO-пінів
- ROM: 384 КБ (використовується внутрішньо, не доступний для користувача)
- 4 шини SPI (2 з них доступні для користувача)
- 2 шини I2S
- 2 шини I2C
- 3 шини UART
- 8-канальний ШИМ-контролер
- 2 10-канальні 12-бітні АЦП
- Апаратний криптографічний прискорювач (AES, RSA, RNG, HMAC)
- USB-контролер (з підтримкою USB-OTG та CDC, що дозволяє програмувати мікроконтролер через USB без додаткового UART-перетворювача)

Модуль ESP32-S3-WROOM-1-N16R8 містить в собі деякі додаткові компоненти:

- Суфікс N16 означає наявність 16 MB Flash-пам'яті типу NOR (для зберігання прошивки та вбудованої файлової системи SPIFFS).
- Суфікс R8 означає наявність 8 MB PSRAM (додаткова оперативна пам'ять).
- 36 GPIO-пінів (5 з них використовуються для USB та внутрішньої пам'яті)
- Кристал, необхідний для роботи мікроконтролера
- Антена
- Конденсатори та інші компоненти

Порада: Відмінною властивістю ESP32-S3 є наявність вбудованого мультиплексора GPIO, який дозволяє використовувати апаратний SPI, I2C, I2S, UART та інші шини на будь-яких GPIO-пінах.

2.2 Креслення (V2)

2.2.1 Електронна схема (v2)

- Завантажити PDF: PDF
- Онлайн-перегляд схеми

2.2.2 Креслення PCB (v2)

- Завантажити PDF: PDF
- Онлайн-перегляд друкованої плати

Розділ 3

Як зібрати Лілку

Попередження: Ця документація знаходиться в розробці. Інформація є неповною, може бути недостовірною і/або застарілою, і може значно змінюватися в майбутньому.

Попередження: Будь ласка, уважно прочитайте статті у цьому розділі, перш ніж почати збирати Лілку.

3.1 Комплектації

Попередження: Ця документація знаходиться в розробці. Інформація є неповною, може бути недостовірною і/або застарілою, і може значно змінюватися в майбутньому.

Лілка - це модульний пристрій.

Комплектація залежить від обраного вами варіанту збірки. Перелік компонентів для кожного варіанту збірки вказаний в наступному розділі.

3.1.1 1. Базова (мінімальна) комплектація

Існує основний варіант збірки, який включає в себе:

- Друкована плата «Лілка»
- Мікроконтролер ESP32-S3-WROOM-1-N16R8
- Дисплей 1.69" IPS 240x280 ST7789
- Модуль microSD-карти
- Модуль роз'єму USB Type C
- П'єзо-динамік
- Резистори, конденсатори та регулятор напруги
- Вимикач живлення

> Попередження: Для використання БАЗОВОЇ комплектації вам ПОТРІБНО ЗАМКНУТИ джампер JP1 на друкованій платі, з'єднавши його контакти між собою за допомогою припою.

3.1.2 2. Базова + Батарея

Додатково можна встановити батарею. Для цього потрібно додаткові компоненти:

- Літій-полімерна батарея 3.7 В (на ваш вибір)
- Зарядний модуль TP4056
- Діод 1N4001
- Транзистор IRLML6401 (корпус - SOT-23)
- Декілька додаткових резисторів

> Попередження: Для використання комплектації З БАТАРЕЄЮ вам ПОТРІБНО НЕ ЗАМИКАТИ джампер JP1 на друкованій платі. Якщо ви вже замкнули його, вам потрібно його розімкнути, нагрівши контакти паяльником і відокремивши їх один від одного.

3.1.3 3. Базова + Аудіо

- Звуковий модуль на ваш вибір
- Динамік або гніздо для навушників

3.2 Перелік деталей

Попередження: Ця документація знаходиться в розробці. Інформація є неповною, може бути недостовірною і/або застарілою, і може значно змінюватися в майбутньому.

Зміст

3.2.1 Базовий комплект

Table 1: Базовий комплект

К-сть	ID	Значення	Назва	В Україні	AliExpress	Mouser	Ціна в ₴	Примітка
1	BZ1	Buzzer (2.54mm між контактами)	Passive Buzzer	mini-tech.com.ua	Посилання		5	
3	C1, C2, C3	1uF	1uF Capacitor	m-teh.com.ua	Посилання		9	Можна комплектувати з цього набору
1	J1	~	USB Type-C to DIP Adapter	arduino.ua	Посилання		12	
1	J2	Ext	Pin Header	arduino.ua	Посилання		5	Варіант з AliExpress містить пари тато+мама, тому задовольняє J2 та U2J1
1	J3	uSD	SD Card Interface Module	diyshop.com.ua	Посилання		24	
1	R1	10K	10K Resistor	arduino.ua	Посилання		3	
1	R2	100K	100K Resistor	arduino.ua	Посилання		2	
1	SW1-SW8	Arrows + A/B/C/D	Button Kit (12mm)	• arduino.ua • imrad.c (кнопки) • imrad.c (ковпачки)	Посилання		36	В комплекті від arduino.ua лише 6 шт
2	SW9, SW10	Select/Start	SMD Tactile Button	imrad.com.ua	Посилання		4	
1	SW11	Power	Toggle Switch	arduino.ua	Посилання		10	Кнопка має бути

3.2.2 Компоненти для батареї

Table 2: Компоненти для батареї

К-сть	ID	Значення	Назва	В Україні	AliExpress	Mouser	Ціна в	Примітка
1	D1	1N4001	1N4001 Diode	voron.ua	Посилання		6	
1	J4	TP4056	TP4056 Charging Module	arduino.ua	Посилання		16	Бажано брати варіант з micro USB (займає менше місця, а гніздо USB Type-C вже є на платі Лілки)
1	Q1	IRLML6401 (SOT-23)	IRLML6 MOSFE	• rcscom • imrad.c	Посилання	Посилання	4	
1	R3	100K	100K Resistor	arduino.ua	Посилання		2	
1	R4	33K	33K Resistor	justas-electronics.co	Посилання		2.1	
1	-	Bat	Будь-яка LiPo на ваш розсуд	m-teh.com.ua			116	
Разом								

3.2.3 Компоненти для звуку

Table 3: Компоненти для звуку

К-сть	ID	Значення	Назва	В Україні	AliExpress	Mouser	Ціна в	Примітка
1	J5	MAX98357	Mono Audio Amplifier Module	arduino.ua	Посилання	Посилання	92	
1	-	Speaker	8 Ohm 1 W Speaker	arduino.ua	Посилання		17	Або гніздо (див. наст. пункт), або будь-який інший динамік на ваш розсуд
1	-	Audio Jack (mono)	3.5mm Audio Jack	imrad.com.ua	Посилання		8	Лише якщо ви вирішили використовувати навушники замість динаміка
Разом								

3.2.4 Не входять в жодний комплект

- MicroSD карта
- USB Type-C кабель

3.3 Що робить кожен компонент?

Якщо ви не знайомі з радіоелектронікою, вам, можливо, цікаво: навіщо потрібен кожен компонент Лілки?

Давайте розглянемо всі компоненти, з яких складається Лілка, і їхні основні функції.

3.3.1 Мікроконтролер ESP32-S3-WROOM-N16R8 (U1)

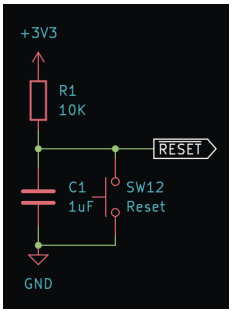
Мікроконтролер ESP32-S3-WROOM-N16R8 від компанії Espressif - це мозок Лілки.

Він відповідає за управління всіма іншими компонентами, зчитування даних з датчиків, керування виводами, роботу з Wi-Fi і Bluetooth, і багато іншого.

Саме він місить флеш-пам'ять, в яку записується прошивка Лілки, і вбудовану пам'ять RAM.

Детальніше про мікроконтролер можна прочитати тут: Мікроконтролер.

3.3.2 Конденсатор C1, резистор R1 та кнопка SW12 («Reset»)



Ці компоненти відповідають за увімкнення мікроконтролера та його перезавантаження.

Річ у тім, що при під'єднанні живлення до Лілки, мікроконтролер не буде працювати коректно, доки не відбудеться процедура скидання (reset).

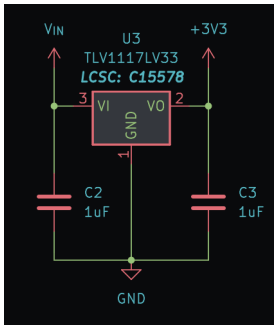
Уявіть собі, що сигнал скидання (reset) - це наче кава: коли на мікроконтролер подається живлення, він «прокидається» і починає спросоння робити неадекватні речі. Сигнал reset - це як кава: він «продупляє» мікроконтролер, і той починає працювати коректно.

Тобто скидання - це процес, який переводить мікроконтролер у початковий стан, в якому він може розпочати виконання програми з самого початку.

Резистор R1 і конденсатор C1 утворюють так зване RC-коло, або «коливальний контур», який відповідає за затримку сигналу скидання. Це означає, що при підключенні живлення до Лілки, мікроконтролер розпочне роботу не відразу, а через деякий час (долю мікросекунди).

Цей коливальний контур є абсолютно необхідним і його потребують майже всі мікроконтролери: без нього мікроконтролер може працювати некоректно або взагалі не запускатися.

3.3.3 Регулятор напруги U3 та конденсатори C2 і C3



Ці компоненти відповідають за стабілізацію напруги живлення Лілки.

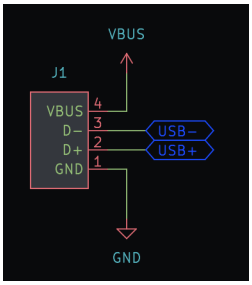
Мікроконтролер ESP32-S3-WROOM-N16R8 працює при напрузі від 3.0V до 3.6V. Але:

- Живлення від USB-порту комп'ютера може бути від 4.75V до 5.25V.
- Живлення від акумулятора може бути від 2.4V до 4.2V.

Обидві з цих напруг майже стовідсотково спалять мікроконтролер! Для цього і потрібен регулятор напруги U3: він перетворює вхідну напругу в напругу 3.3V, яка потрібна мікроконтролеру.

Конденсатори C2 і C3 відповідають за стабілізацію напруги: вони зберігають енергію, яка потрібна компонентам Лілки, коли напруга від живлення змінюється. Рекомендовані номінали цих конденсаторів переважно вказані в документації регулятора напруги.

3.3.4 Роз'єм USB (J1)



Модуль J1 - це всього лиш плата з припаяним роз'ємом USB Type C. Вона не містить жодних інших компонентів - лише роз'єм.

Паяти роз'єм USB Type C безпосередньо на головну плату Лілки було б дуже складно: роз'єм має дуже маленькі контакти, які важко паяти. Саме тому ми і використовуємо модуль J1: він має великі контакти, які можна легко паяти на основну плату Лілки.

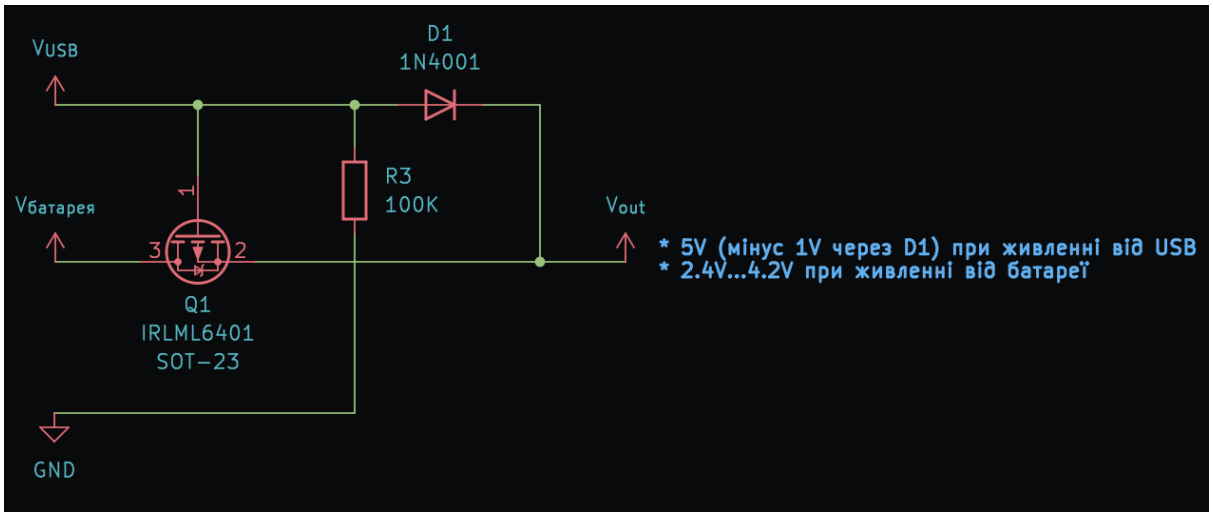
3.3.5 Модуль заряду-розряду LiPo-акумулятора (J4)

Якщо ви збираєте комплект Лілки з акумулятором, цей модуль відповідає за зарядку акумулятора від USB-порту, а також забезпечує захист від перезаряду та перерозряду акумулятора при роботі Лілки від акумулятора.

Він працює на основі мікросхеми TP4056, тому його часто так і називають - «модуль TP4056».

На виході цей модуль дає напругу, яка дорівнює напрузі акумулятора, тобто від 2.4V до 4.2V. Ця напруга подається на регулятор напруги U3, який перетворює її в 3.3V для живлення мікроконтролера.

3.3.6 Польовий транзистор P-типу Q1, резистор R3 та діод D1



Ці компоненти потрібні для коректної роботи Лілки, коли ви одночасно заряджаєте акумулятор і живите Лілку від USB-порту.

Річ у тім, що одночасно заряджати акумулятор від USB і в той же час живити Лілку від цього ж акумулятора - небезпечно: модуль TP4056 (J4) не вміє коректно відслідковувати стан заряду акумулятора, якщо той використовується Лілкою, і може подати на нього невідповідну напругу.

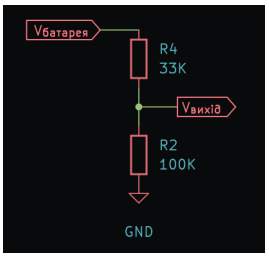
Тому ми використовуємо польовий транзистор P-типу (Q1), який «вимикається» при підключенні живлення від USB, і таким чином при під'єднаному USB-живленні Лілка живиться від USB, а не від акумулятора.

Резистор R3 - це слабка підтяжка польового транзистора Q1 до землі, щоб при відсутності напруги від USB цей транзистор не вимикався сам по собі, а за замовчуванням був увімкнений, щоб Лілка живилася від акумулятора.

Діод D1 - це захисний діод, який не дозволяє напрузі від акумулятора подаватися на USB-порт, коли USB-живлення під'єднане. Це потрібно, щоб не пошкодити USB-порт комп'ютера

або зарядний пристрій. Крім цього, як бонус - він частоково знижує напругу (на ~0.7V), і тому регулятор напруги U3 при живленні Лілки від USB буде трохи менше нагріватися.

3.3.7 Резистори R2 і R4



Ці резистори здійснюють ділення напруги для вимірювання напруги акумулятора.

Напруга акумулятора може бути в діапазоні від 2.4V до 4.2V, але АЦП (аналого-цифровий перетворювач) мікроконтролера ESP32-S3-WROOM-N16R8 вміє вимірювати напругу лише від 0V до 3.1V.

Тому ми використовуємо резистори R2 і R4, які здійснюють ділення напруги, використовуючи формулу:

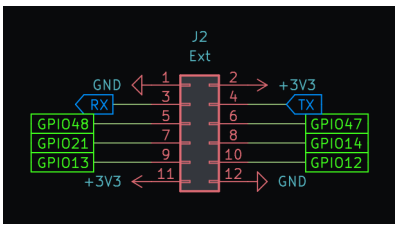
$$V_{\text{вихід}} = V_{\text{батарея}} \times \left(\frac{R_4}{R_2 + R_4} \right)$$

Номінали резисторів R2 і R4 вибираються так, щоб напруга на виході ділення була 3.1V при вхідній напрузі 4.2V.

В нашому випадку ми використовуємо резистори R2 = 100 кОм і R4 = 33 кОм.

Ми могли б використати і резистори менших номіналів - скажімо, 10 кОм і 3.3 кОм, але вони б витрачали в 100 разів більше енергії та швидше розряджали акумулятор.

3.3.8 Роз'єм розширення (J2)

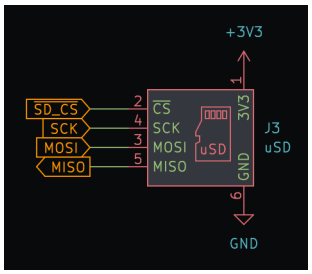


Лілка - це набір для розробки, і ви, можливо, захочете підключити до неї додаткові датчики, дисплеї, камери, акселеометри, сервоприводи чи ще щось.

Для цього і потрібен роз'єм розширення J2: він має виводи для живлення сторонніх компонентів, а також декілька GPIO-ліній мікроконтролера, які можна використовувати для підключення додаткових пристроїв.

Детальніше про роз'єм розширення можна прочитати в розділі Роз'єм розширення.

3.3.9 Модуль SD-карти (J3)

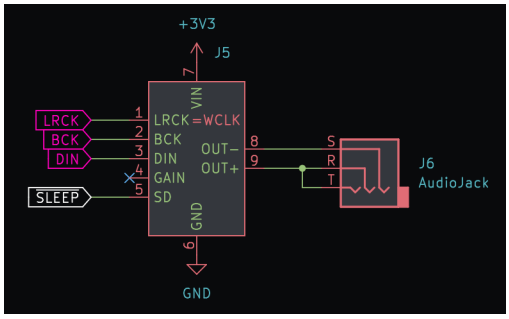


Цей модуль дозволяє Лілці читати та зберігати дані на SD-карті, а також запускати з неї сторонні прошивки.

SD-карти дозволяють працювати з ними через шину SPI, яка підтримується мікроконтролером ESP32-S3-WROOM-N16R8.

Більше інформації є тут: Карта microSD.

3.3.10 ЦАП (DAC) - Цифро-аналоговий перетворювач (J5)



Цей компонент відповідає за генерацію аналогових сигналів.

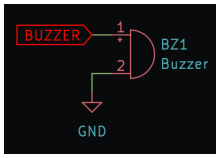
Найпоширеніше використання ЦАП - генерація звуків: відтворення мелодій, голосів, звуків природи, і т.д.

Лілка використовує модуль ЦАП на базі мікросхеми MAX98357A від компанії Maxim Integrated.

Це дозволяє Лілці відтворювати звуки високої якості, а також підключати до неї аудіопристрої виведення звуку - наприклад, колонки, навушники, аудіо-підсилювачі тощо.

Для роботи з ЦАПом використовується шина I2S.

3.3.11 П'єзо-динамік (BZ1)



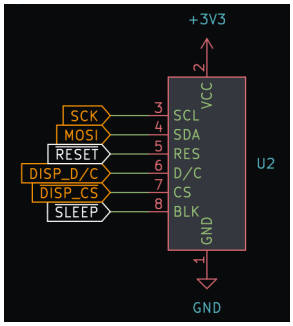
П'єзо-динамік - це простий динамік, який може відтворювати звуки. Його можна під'єднувати безпосередньо до GPIO-виводів мікроконтролера.

Основні плюси п'єзо-динаміка - його простота та ціна. Він не потребує жодних додаткових компонентів: просто підключіть його до GPIO-виводів мікроконтролера і відтворюйте звуки.

Головний мінус - обмежені можливості. П'єзо-динамік може відтворювати лише прості звуки і не здатний відтворювати музику високої якості. Крім того, ESP32-S3 не має вбудованого цифро-аналогового перетворювача, тому звукова хвиля може мати лише форму меандру (т. зв. «прямокутна» звукова хвиля).

Якщо ви хочете відтворювати музику високої якості, вам краще використати додатковий модуль ЦАП (див. вище).

3.3.12 IPS TFT дисплей (U2)



Дисплей. Куди ж без нього? ;)

Цей дисплей використовує IPS-матрицю, яка забезпечує високу якість зображення, широкі кути огляду, яскравість та насиченість кольорів.

Лілка використовує дисплей на базі мікросхеми ST7789 від компанії Sitronix.

3.3.13 Кнопки

Думаю, тут все зрозуміло: кнопки використовуються для керування Лілкою.

Кожна кнопка підключена до свого GPIO-виводу мікроконтролера і відповідає за певну функцію.

Примітка: «Але чому ви не використали, скажімо, регістр зсуву для підключення кнопок, чи розширювачі I/O?» - запитаєте ви. «Ви могли б підключити до мікроконтролера 8 кнопок, використовуючи всього 3 GPIO-виводи!»

Відповідь проста: мета Лілки - бути простою у збірці. Регістри зсуву - це додаткові компоненти, які вам потрібно паяти і які займають місце на і так невеликій платі Лілки, тому ми вирішили обійтися без них.

3.4 Збір пристрою

PCB Лілки спроєктована для максимально простого збору зі збереженням її мінімального розміру.

Більшість компонентів - це THT-компоненти, тобто компоненти, які вставляються в отвори плати і паяються з задньої сторони. Це дозволяє збирати плату без використання спеціального обладнання.

Декілька компонентів - це SMD-компоненти, які паяються зверху. Серед них - мікроконтролер ESP32-S3-WROOM-1-N16R8, регулятор напруги TLV1117LV-33DCYR та транзистор IRLML6401.

3.4.1 Порядок збору

1. Перш за все, припаяйте SMD-компоненти.

   Вони займають найменше місця та не заважатимуть вам при пайці решти компонентів.**

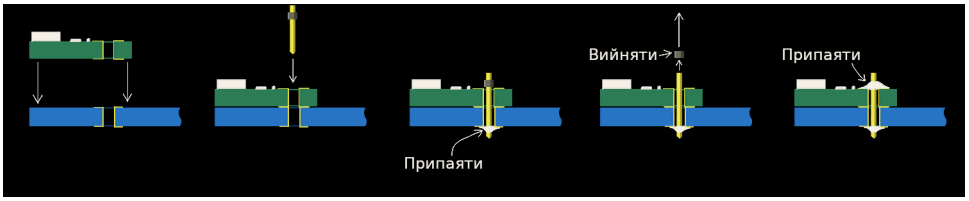
   - U1 (мікроконтролер ESP32-S3-WROOM-1-N16R8)
   - U3 (регулятор TLV1117LV-33DCYR)
   - Q1 IRLML6401 (якщо ви використовуєте комплектацію з батареєю)

2. Припаяйте модуль USB-C (J1).

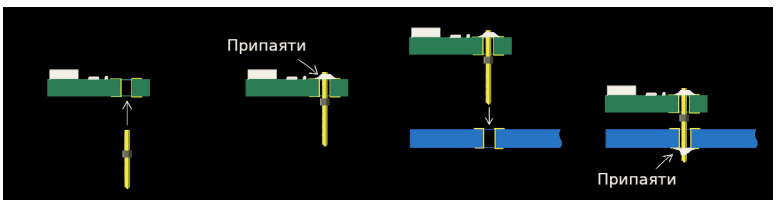
   Переконайтесь, що ви вставили модуль USB-C вірно. Він має бути вирівняний з краєм плати.

   Є два способи паяти такі модулі на плату:

   - Спосіб 1: Ви можете прикласти його до плати та вставити штирьові контакти крізь нього та плату, утворивши таким чином «сендвіч». Після цього вам потрібно припаяти штирьові контакти з обох сторін. Таким чином модуль щільно прилягатиме до плати та не буде провисати.

   - Спосіб 2: Ви можете спершу припаяти штирьові контакти на модуль USB-C, а потім припаяти модуль на плату. Цей спосіб простіший, але модуль буде провисати, оскільки між ним та платою буде вільний простір, утворений пластиковим тримачем штирьових контактів.

   Є й інші способи - не бійтесь проявляти креативність!

3. Якщо ви збираєте комплектацію з батареєю, припаяйте наступні компоненти (вони всі позначені пунктирними лініями на задній стороні плати)

    - Q1 IRLML6401
    - D1 (діод 1N4001)
    - Резистори R3 (100 кОм) та R4 (33 кОм)
    - J4 (модуль заряду/розряду батареї на базі TP4056)
    - LiPo батарея на ваш вибір (припаюється безпосередньо до пінів модуля заряду/розряду, позначених як B+ та B-)

    Якщо ви збираєте комплектацію без батареї, не паяйте ці компоненти!

    Натомість замкніть джампер JP1, який знаходиться всередині посадкового місця діода D1. Для цього використайте паяльник та припій, щоб замкнути контакти джампера.

4. Припаяйте наступні компоненти:

    - Резистори R1 (10 кОм) R2 (100 кОм)
    - Конденсатори C1, C2 та C3 (1 мкФ)

5. Припаяйте 6-мм тактильні кнопки:

    - SW9 (Select)
    - SW10 (Start)
    - SW12 (Start)

6. Припаяйте модуль SD-карти J3.

    Як і модуль USB-C, модуль SD-карти можна припаяти двома способами.

    Переконайтесь, що ви вставили модуль SD-карти вірно! Металеве гніздо для SD-карти повинне знаходитися зовні плати!

7. Припаяйте 8 12-мм кнопок:

    - SW1 (Up)
    - SW2 (Down)
    - SW3 (Left)
    - SW4 (Right)
    - SW5 (A)
    - SW6 (B)
    - SW7 (C)
    - SW8 (D)

8. Якщо ви збираєте комплектацію зі звуком, припаяйте ЦАП-модуль U4 (MAX98357A).

    - Якщо ви плануєте використовувати навушники, спершу припаяйте роз'єм для навушників J6 до плати. Після цього обов'язково припаяйте всі 9 контактів модуля MAX98357A.

- Якщо ж ви хочете використовувати власний динамік, вам достатньо припаяти лише 7 основних контактів, а контакти динаміка припаяти безпосередньо до пінів (+) та (-) модуля.

9. Припаяйте п'єзоелектричний динамік BZ1.

   Він дозволяє відтворювати примітивні звуки. Він працює окремо від ЦАП-модуля, тому ви можете використовувати його навіть якщо не паяєте ЦАП-модуль.

10. Припаяйте роз'єм дисплея та роз'єм розширення:

- U2 (роз'єм дисплея типу «мама»)
- J2 (роз'єм розширення типу «тато»)

11. Припаяйте вимикач живлення SW11.

Вітаємо, ваша плата готова до використання! Тепер вам залишається лише завантажити програмне забезпечення на ESP32-S3. Для цього читайте наступний розділ - Програмування.

Розділ 4

Програмування

Отже, ви зібрали Лілку!

В цьому розділі ви знайдете інструкції, як налаштувати робоче середовище, як її запрограмувати, як розширити функціонал, а також як розробляти власні розширення.

4.1 Налаштування середовища розробки

Отже, ви зібрали Лілку. Чи, можливо, якийсь інший пристрій на базі мікроконтролера ESP32. І ви хочете написати для нього програмне забезпечення. З чого почати? (Саме так: багато що з цього документу може бути корисним і для інших пристроїв на базі ESP32, але ми будемо говорити здебільшого саме про Лілку.)

Для розробки програмного забезпечення для Лілки ми використовуємо середовище PlatformIO, і дуже рекомендуємо вам теж спробувати його. До речі, PlatformIO - це український проект, і ми пишаємося тим, що він став таким популярним у всьому світі!

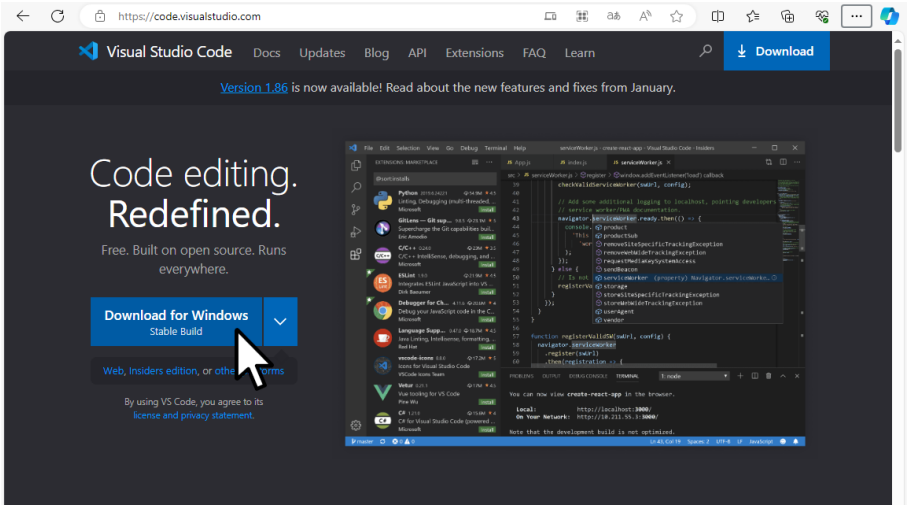
PlatformIO - це набір інструментів для розробки вбудованого програмного забезпечення, який автоматизує процес збірки, завантаження та налагодження програмного забезпечення для мікроконтролерів. Якщо ви колись працювали з Arduino IDE, то ви знаєте, що воно хоч і зручне, але має свої обмеження. PlatformIO - це таке собі «Arduino для дорослих».

PlatformIO автоматично встановлює необхідні залежності, такі як компілятори, бібліотеки та інші інструменти, що необхідні для написання програм.

В цьому документі описано процес налаштування середовища розробки для Лілки з використанням PlatformIO та Visual Studio Code для ОС Windows.

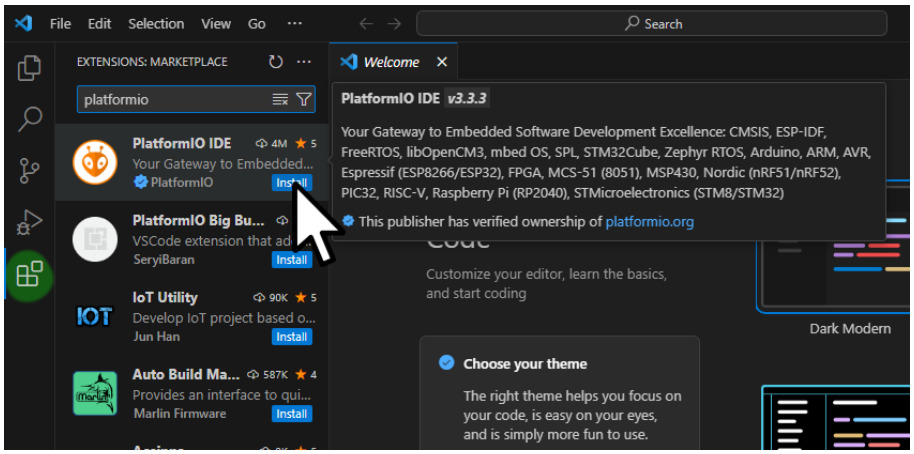
Якщо ви використовуєте MacOS, процес буде схожим, але можуть бути деякі відмінності в налаштуваннях. А якщо ви використовуєте GNU/Linux, то ви, скоріш за все, вже й так знаєте, що робити, і цей документ вам не потрібен. ;)

1. Встановіть Visual Studio Code з офіційного сайту: https://code.visualstudio.com/



2. Запустіть Visual Studio Code та встановіть плагін PlatformIO IDE.

   Для цього відкрийте вкладку Extensions (Ctrl + Shift + X), введіть «PlatformIO» в поле пошуку та встановіть плагін.

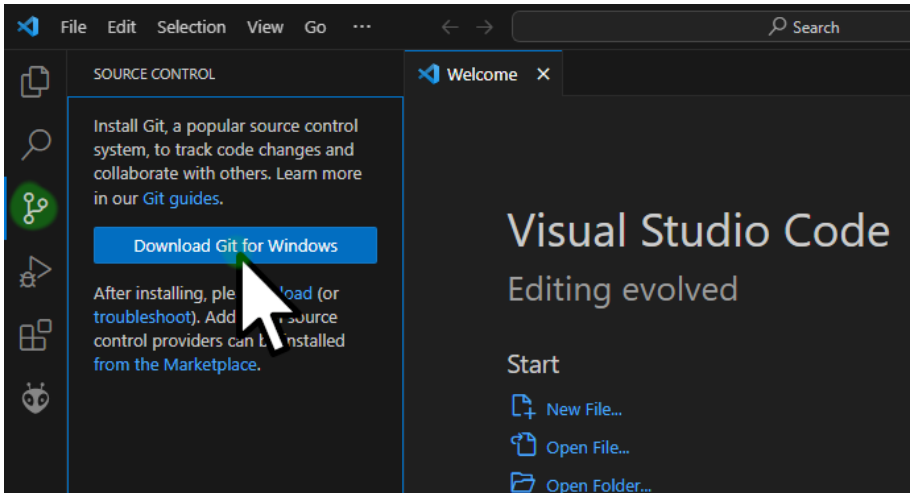


3. Тепер нам потрібно встановити Git.

   Git - це система керування версіями, яка дозволяє ділитися кодом з іншими розробниками та використовувати код з відкритих репозиторіїв.

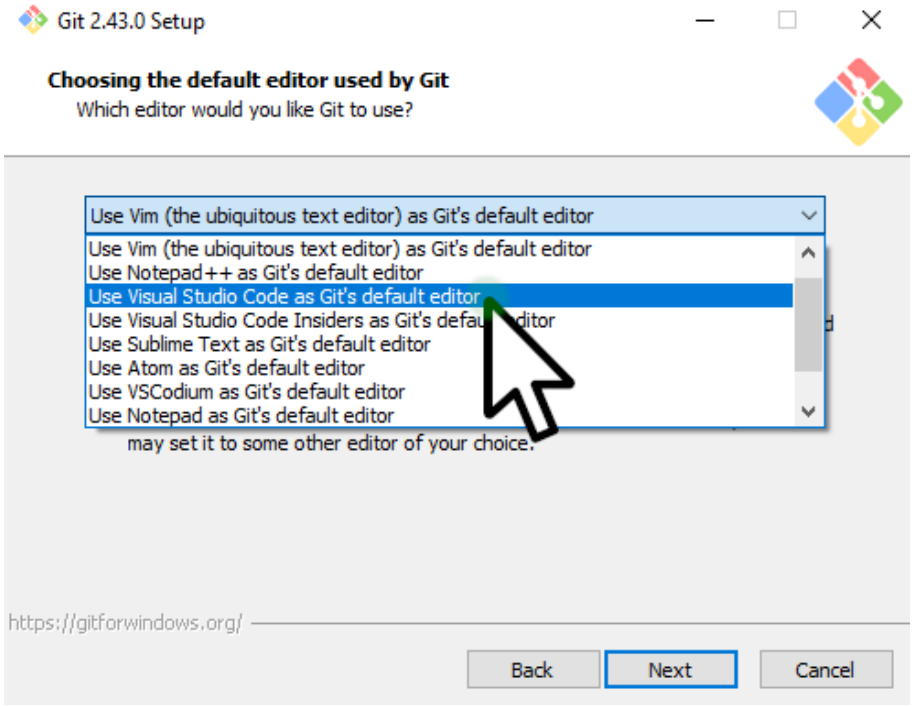
   Весь код для Лілки, а також бібліотеки, які використовуються в проєкті, знаходяться на GitHub, тому нам потрібно встановити Git, щоб здійснювати роботу з репозиторіями.

   Для встановлення Git перейдіть на вкладку Source Control (Ctrl + Shift + G) та натисніть «Download Git for Windows».



В процесі встановлення не змінюйте жодних параметрів, залиште все так, як є за замовчуванням, окрім кроку «Choosing the default editor used by Git».

На цьому кроці виберіть «Use Visual Studio Code as Git's default editor».



4. Після встановлення Git перезапустіть Visual Studio Code.

Тепер ви можете компілювати та завантажувати програми для Лілки, а також працювати з репозиторіями на GitHub.

4.2 Keira OS

Ми написали для Лілки власну прошивку, яка називається «Keira OS» (Операційна Система «Кіра») і включає в себе основні функції для демонстрації можливостей пристрою.

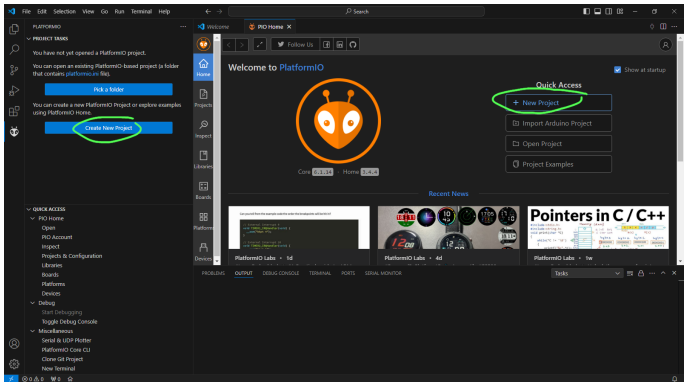
Keira - це операційна система, що базується на проєкті FreeRTOS. Вона підтримує мультизадачність, мережу, запуск Lua-програм та сторонніх прошивок з SD-карти та багато іншого.

Більше інформації про Keira OS - на сторінці Можливості.

Якщо вам кортить якнайшвидше спробувати Keira OS, переходьте на сторінку Запуск.

4.3 Створення власної прошивки

1. Відкрийте Visual Studio Code, перейдіть на вкладку PlatformIO та виберіть Create New Project.

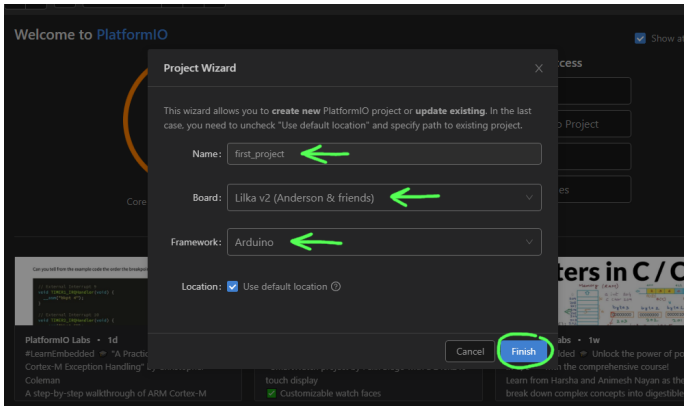


   У вікні, що відкриється, натисніть на кнопку New Project.

2. Придумайте назву свого проєкту - наприклад, first_project.

   У переліку boards оберіть Lilka v2 (Anderson & friends).

   У переліку frameworks оберіть Arduino.



   Після цього натисніть на кнопку Finish.

Примітка: Якщо ви не бачите в переліку Lilka v2 (Anderson & friends) або у вас виникає помилка при створенні проєкту, вам потрібно оновити пакети PlatformIO.

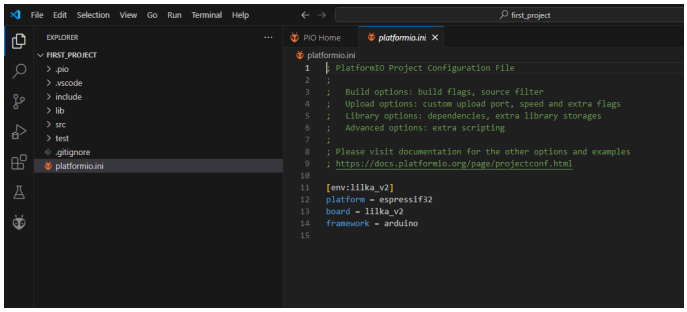
Для цього виконайте команду pio pkg update -g -p espressif32 в терміналі Visual Studio Code.

Щоб відкрити термінал, в панелі Quick Access виберіть Miscellaneous і тоді натисніть на PlatformIO Core CLI. Консоль відкриється внизу вікна Visual Studio Code.

Після цього вам слід перезапустити Visual Studio Code.

---

3. Ваш новий проєкт відкриється у новому вікні Visual Studio Code, і ви побачите файл platformio.ini.

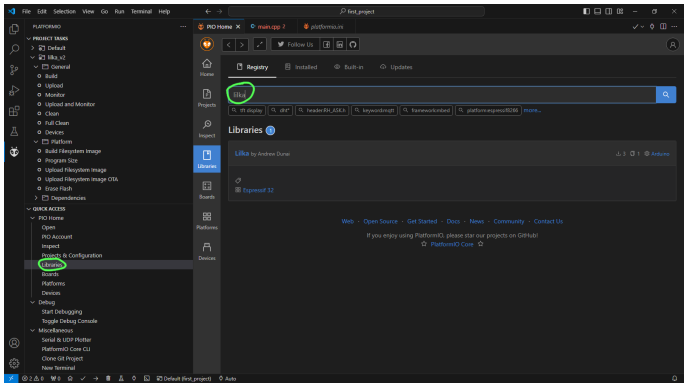


Це - конфігураційний файл проєкту, в якому вказані всі налаштування проєкту: платформа, фреймворк, бібліотеки тощо.

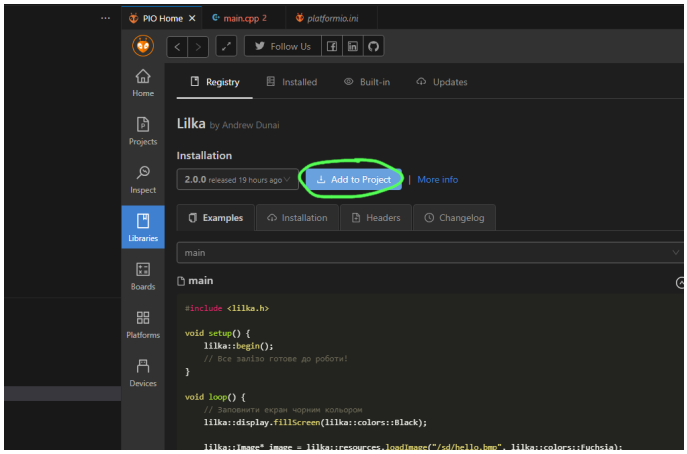
Ми можете редагувати його вручну, а можете використовувати графічний інтерфейс PlatformIO.

4. Давайте додамо до проєкту бібліотеку lilka. Для цього відкрийте вкладку PlatformIO та в панелі Quick Access виберіть Libraries.

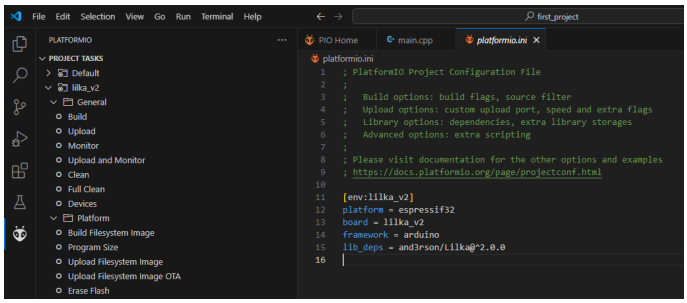
Після цього введіть у поле пошуку lilka:



Тепер натисніть на кнопку Add to Project.



Після завершення завантаження бібліотеки ви побачите, що вона з'явилась у полі lib_deps файлу platformio.ini.



Це поле вказує на те, які бібліотеки використовуються у проєкті. PlatformIO автоматично завантажує та встановлює всі бібліотеки, вказані у цьому полі, а також їх залежності.

Встановлення бібліотеки lilka автоматично встановлює різні бібліотеки для роботи з Лілкою, наприклад Arduino-GFX, яка використовується для роботи з дисплеєм Лілки.

5. Напишемо простий код нашої прошивки. Відкрийте файл src/main.cpp та напишіть наступний код:

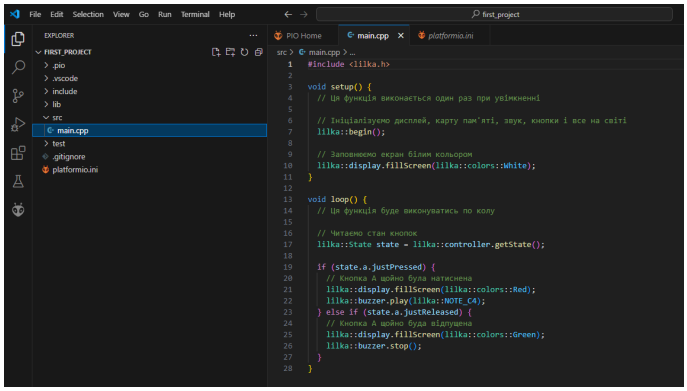
```
#include <lilka.h>

void setup() {
  // Ця функція виконається один раз при увімкненні

  // Ініціалізуємо дисплей, карту пам'яті, звук, кнопки і все на світі
  lilka::begin();

  // Заповнюємо екран білим кольором
  lilka::display.fillScreen(lilka::colors::White);
}

void loop() {
  // Ця функція буде виконуватись по колу

  // Читаємо стан кнопок
  lilka::State state = lilka::controller.getState();
```

(continues on next page)

(continued from previous page)

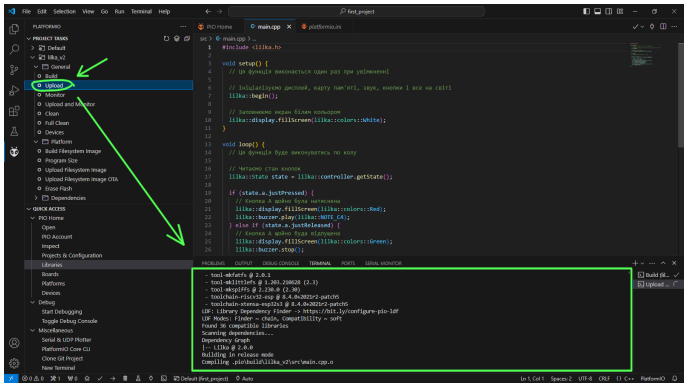
```
  if (state.a.justPressed) {
    // Кнопка A щойно була натиснена
    lilka::display.fillScreen(lilka::colors::Red);
    lilka::buzzer.play(lilka::NOTE_C4);
  } else if (state.a.justReleased) {
    // Кнопка A щойно буда відпущена
    lilka::display.fillScreen(lilka::colors::Green);
    lilka::buzzer.stop();
  }
}
```

Це - проста програма, яка при натисканні на кнопку A змінює колір екрану на червоний та відтворює звук, а при відпусканні - заповнює екран зеленим кольором та зупиняє звук.



6. Тепер ми можемо скомпілювати нашу прошивку та завантажити її в Лілку.

   Для цього під'єднайте Лілку до комп'ютера за допомогою USB-кабеля, відкрийте вкладку PlatformIO та натисніть на дію Upload.



Після завершення компіляції та завантаження прошивки ви побачите, як Лілка виконує вашу програму. Спробуйте натиснути та відпустити кнопку A та переконайтеся, що все працює!

4.4 Роз'єм розширення

Зміст

4.4.1 Загальна інформація

Роз'єм розширення (англ. extension header) - це роз'єм, який дозволяє підключати до основної плати додаткові модулі, наприклад датчики, дисплеї, сервоприводи тощо.

Лілка має 12-контактний роз'єм розширення.

Ці піни можна використовувати для будь-якої цілі. Саме тому вони і називаються GPIO (General Purpose Input/Output) - вхід/вихід загального призначення.

На роз'єм розширення Лілки виведені наступні піни:

- 3.3V - живлення 3.3 Вольти
- GND - земля
- 12, 13, 14, 21, 47, 48 - GPIO
- TX, RX - додатковий UART. Ці піни можна використати для з'єднання Лілки з комп'ютером через перетворювач USB-UART та навіть прошивати її через цей порт. Але ви також можете без проблем використовувати ці піни для будь-яких інших цілей в додачу до перелічених вище пінів, а для прошивання використовувати USB-порт Лілки.

Вони не використовуються Лілкою для жодних внутрішніх потреб, тому ви можете використовувати їх для будь-яких цілей.

Завдяки вбудованому мультиплексору, кожен пін можна використовувати для шин I2C, SPI, UART, або просто як вхід/вихід.

Крім того, піни 12, 13 та 14 під'єднані до АЦП (аналого-цифрового перетворювача, ADC):

- 12 - ADC2, канал 1
- 13 - ADC2, канал 2
- 14 - ADC2, канал 3

4.4.2 Приклад використання GPIO

Найпростіший приклад використання GPIO - це підключити до роз'єму розширення світлодіод та резистор, щоб світлодіод світився, коли на пін подаватиметься напруга.

Наприклад, пін 12 можна підключити до анода світлодіода (довший вивід), а катод світлодіода (коротший вивід) - через резистор до землі (наприклад, 100 Ом).

Обчислити значення резистора для світлодіода можна за цим посиланням.

Після цього можна використати наступний код для того, щоб світлодіод блимав:

```
#include <lilka.h>

void setup() {
    lilka::begin();
}

void loop() {
    digitalWrite(12, HIGH);
    delay(500);
    digitalWrite(12, LOW);
    delay(5000);
}
```

4.4.3 Використання GPIO для шини I2C

Щоб використати піни розширення для шини I2C, використайте наступний код:

```
#include <lilka.h>
#include <Wire.h>

#define SDA_PIN 13
#define SCL_PIN 14

void setup() {
    lilka::begin();
    Wire.begin(SDA_PIN, SCL_PIN);
}

void loop() {
    // Почати передачу на адресу 0x42
    Wire.beginTransmission(0x42);
    // Надіслати байт 66
    Wire.write(0x42);
    // Завершити передачу
    Wire.endTransmission();
    delay(1000);
}
```

4.4.4 Використання GPIO для шини SPI

Щоб використати піни розширення для шини SPI, використайте наступний код:

```
#include <lilka.h>

#define SCK_PIN 12
#define MISO_PIN 13
#define MOSI_PIN 14
#define SS_PIN 21

void setup() {
    lilka::begin();

    // Ми використовуємо SPI2, тому що SPI1 використовується для внутрішніх потреб Лілки (для дисплея та SD-картки)
    lilka::SPI2.begin(SCK_PIN, MISO_PIN, MOSI_PIN, SS_PIN);
}

void loop() {
    // Починаємо транзакцію: швидкість - 1 МГц, порядок байтів - MSB, режим SPI - 0
    lilka::SPI2.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));
    // Активуємо пристрій
    digitalWrite(SS_PIN, LOW);
    // Надсилаємо байт 66
    lilka::SPI2.transfer(0x42);
    // Деактивуємо пристрій
    digitalWrite(SS_PIN, HIGH);
    // Завершуємо транзакцію
    lilka::SPI2.endTransaction();
    delay(1000);
}
```

Розділ 5

Keira OS

Ми написали для Лілки власну прошивку, яка називається «Keira OS» (Операційна Система «Кіра») і включає в себе основні функції для демонстрації можливостей пристрою.

Keira - це операційна система, що базується на проєкті FreeRTOS. Вона підтримує мультизадачність, мережу, запуск Lua-програм та сторонніх прошивок з SD-карти та багато іншого.

Більше інформації про Keira OS - на наступній сторінці.

5.1 Можливості

5.1.1 Запуск прошивок з SD-картки

Keira підтримує запуск другорядних прошивок з SD-картки. Такі файли мають розширення .bin. Це дозволяє вам скомпілювати (або завантажити з інтернету) нестандартну прошивку, скопіювати її на SD-картку, і запустити її на Лілці.

Це має деякі великі переваги:

- Ви можете використовувати більше пам'яті, ніж доступно в Keira.
- Вам не потрібно щоразу перепрошивати Лілку, щоб випробувати другорядні прошивки: просто скопіюйте їх на SD-картку.

- Ви можете ділитися своїми скомпільованими прошивками (у вигляді .bin-файлів) з іншими користувачами Лілки, без необхідності відправляти їм весь код вашої власної прошивки.

Найяскравіший приклад використання цієї функції - запуск Doom. Достатньо скопіювати на SD-карту файли doom.bin і doom.wad (або doom1.wad), і ви зможете грати в Doom!

Примітка: Щоб отримати doom.bin, вам потрібно скомпілювати прошивку, що знаходиться в папці firmware/doom, і тоді скопіювати файл doom.bin на SD-картку.

Примітка: Щоб отримати doom.wad, ви можете завантажити безкоштовну shareware-версію Doom за цим посиланням, або ж мати повну ліцензійну копію Doom і витягти з неї файл doom.wad.

Тепер ви можете вставити SD-картку в Лілку і вибрати doom.bin у браузері SD-картки. Лілка перезавантажиться, і ви зможете зіграти в Doom!

Після перезавантаження Лілки ви повернетеся до Keira.

5.1.2 Запуск ігор NES з SD-картки

Keira має вбудовану підтримку запуску ігор NES з SD-картки. Це означає, що ви можете скопіювати файли ігор NES (.rom або .nes) на SD-картку, і грати в них на Лілці.

Для емуляції NES Лілка використовує емулятор Nofrendo. Він може мати деякі обмеження, але, наприклад, Super Mario Bros працює бездоганно.

5.1.3 Запуск Lua-скриптів з SD-картки

Keira має вбудовану підтримку запуску Lua-скриптів з SD-картки. Це означає, що ви можете скопіювати файли Lua (.lua) на SD-картку, і виконувати їх на Лілці.

Для виконання Lua-скриптів Лілка використовує вбудований інтерпретатор Lua, який базується на Lua 5.4.6.

Написати власний Lua-скрипт для Лілки дуже просто. Для цього вам потрібно знати основи Lua, а також використовувати Lua API Лілки. Детальніше про це - в розділі Написання програм на Lua.

Детальніше про те, як писати Lua-скрипти для Лілки, ви можете прочитати в розділі Написання програм на Lua.

5.1.4 Запуск mJS-скриптів з SD-картки

mJS - це мінімалістичний двигун JavaScript, який використовується в Лілці для виконання скриптів. Keira має вбудовану підтримку запуску mJS-скриптів з SD-картки. Це означає, що ви можете скопіювати файли, написані діалектом mJS (з розширенням .js) на SD-картку і виконувати їх на Лілці.

Я не впевнений, наскільки корисною буде ця функція, і ми рекомендуємо використовувати для цього Lua, яка має значно більше можливостей, ніж mJS. Проте така опція є і ви можете спробувати її, якщо вам цікаво.

> Попередження: mJS - це діалект JavaScript, який має свої особливості. Наприклад, він не підтримує класи і має свої власні функції для роботи з рядками, масивами, об'єктами тощо. Тому, якщо ви вирішили використовувати mJS, будьте готові до того, що:
>
> - Вам доведеться вивчати новий діалект JavaScript.
> - Ви не зможете використовувати майже жодну з бібліотек, які ви використовуєте в звичайному JavaScript.
> - Ми не володіємо жодною інформацією про стабільність проєкту mJS загалом.

Додати mJS до Лілки було дуже просто і це дозволяє нам хизуватися тим, що «для Лілки можна писати код на JavaScript!» :)

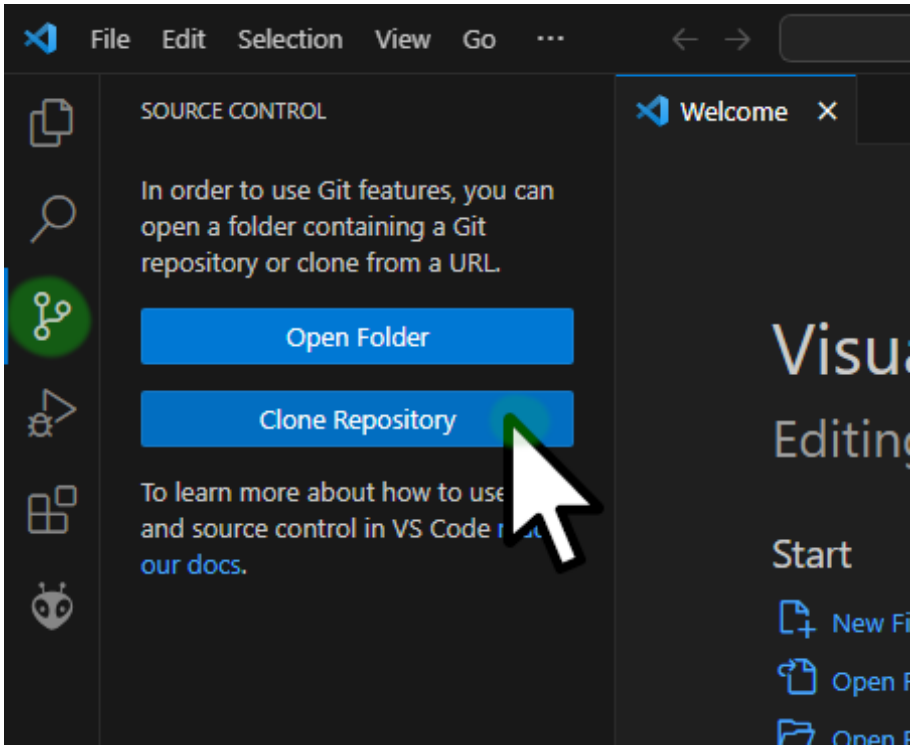
5.2 Запуск

Лілка постачається з готовою прошивкою Keira, яка вміє багато різних речей. Ця прошивка використовується для демонстрації можливостей Лілки, а також як приклад для розробки власних програм.

В цьому розділі ми розглянемо, як завантажити операційну систему Keira в Лілку.

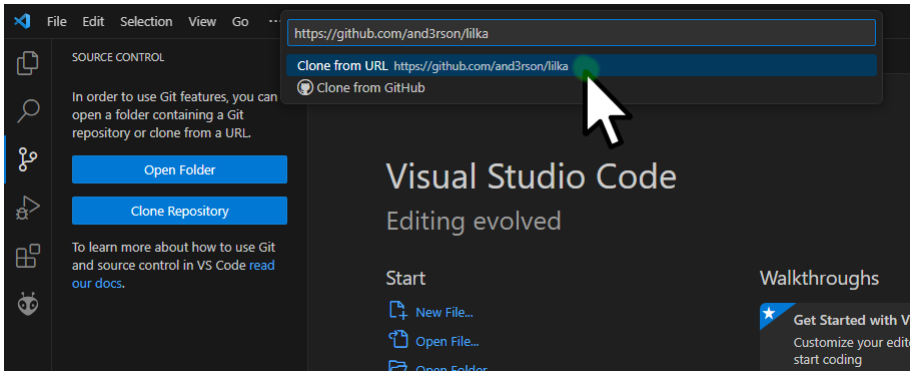
5.2.1 Клонування репозиторію та побудова прошивки

1. Перейдіть на вкладку Source Control (Ctrl + Shift + G) та клонуйте репозиторій Lilka:

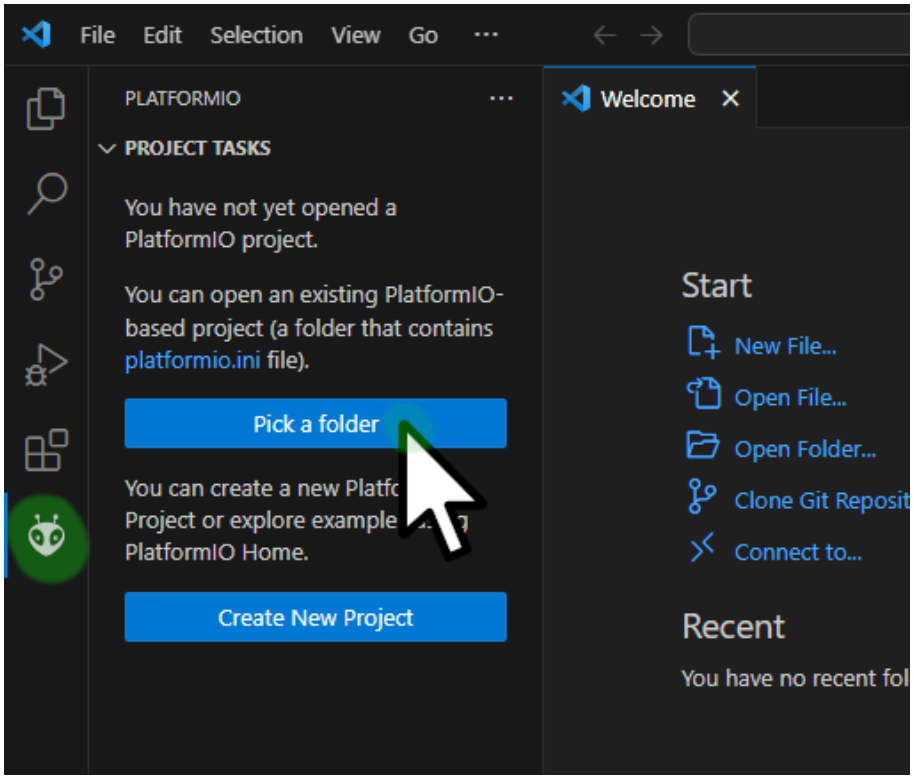


Для цього натисніть «Clone Repository» та введіть адресу репозиторію:

https://github.com/and3rson/lilka

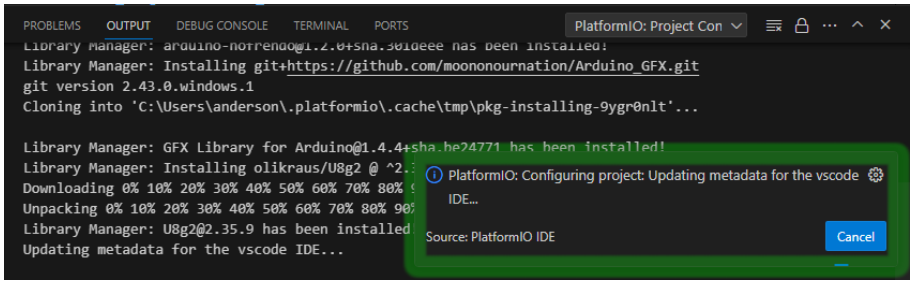


2. Після клонування репозиторію перейдіть на вкладку PlatformIO та імпортуйте проєкт прошивки Keira для Лілки. Для цього натисніть «Open Project» та виберіть директорію lilka/firmware/keira.

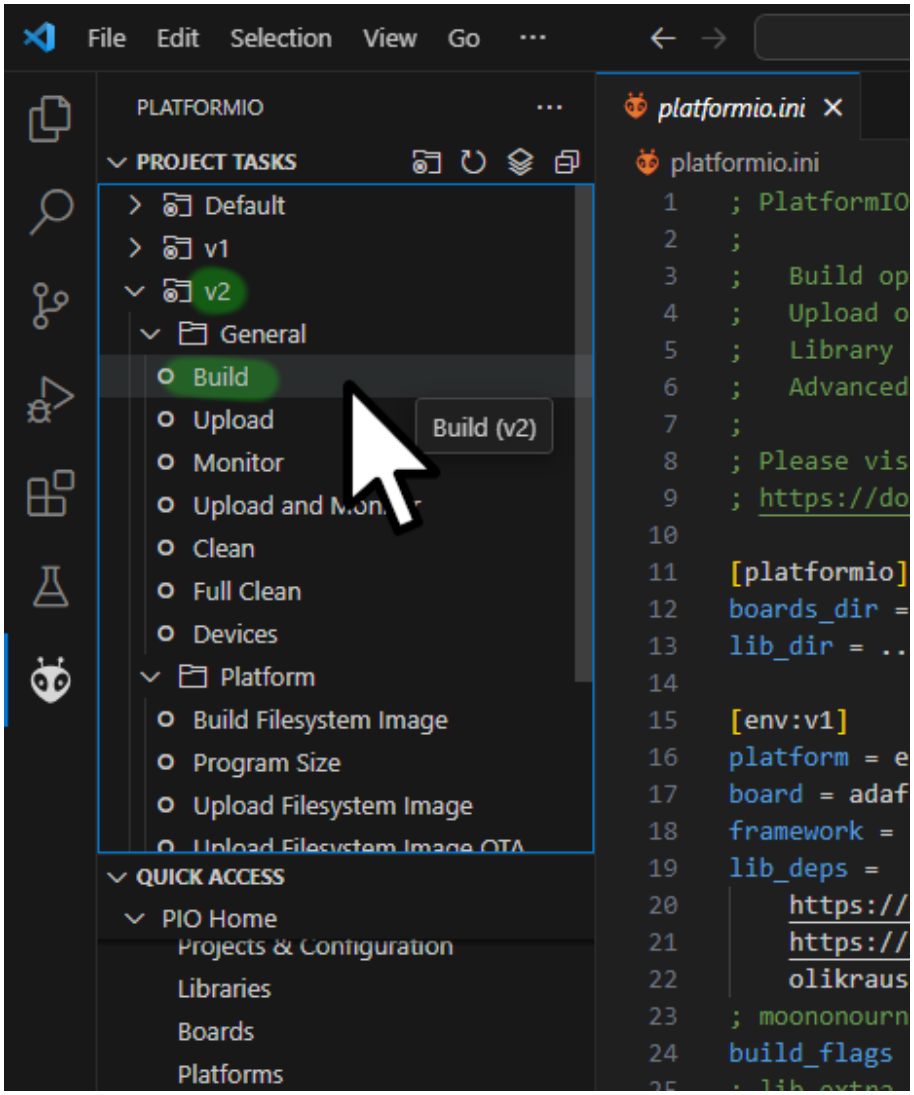


3. Тепер нам потрібно дочекатися налаштування середовища розробки.

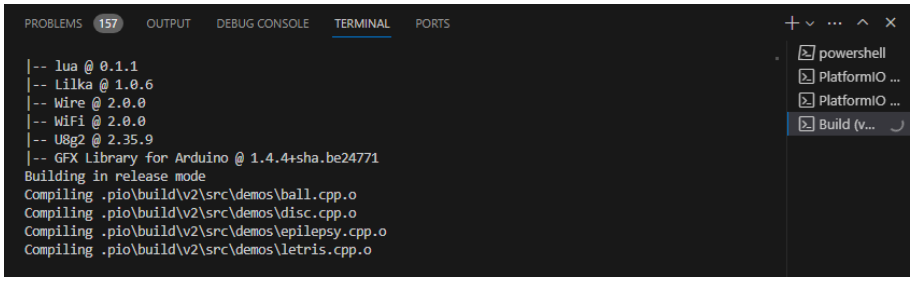
   PlatformIO автоматично встановить необхідні залежності, такі як компілятори, бібліотеки та інші інструменти, що необхідні для написання програм для Лілки.



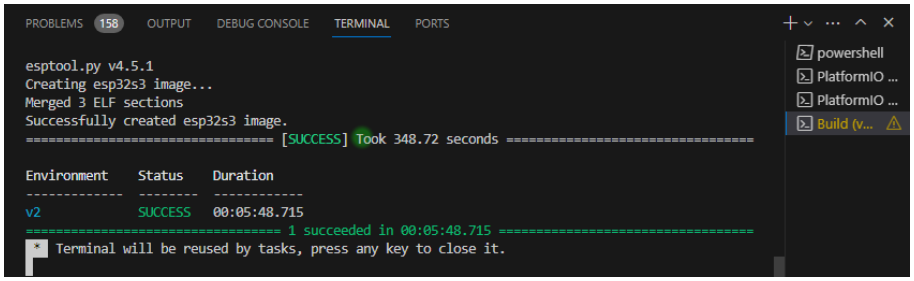
4. Після налаштування середовища розробки перейдіть на вкладку PlatformIO та виберіть v2 в якості цільової платформи. Потім натисніть «Build».



Тепер потрібно дочекатися завершення збірки. Перша збірка може зайняти деякий час.



Щойно збірка завершиться, ви побачите повідомлення про успішне завершення збірки:



5.2.2 Завантаження прошивки в Лілку

1. Підключіть Лілку до комп'ютера за допомогою USB-кабеля та увімкніть її.

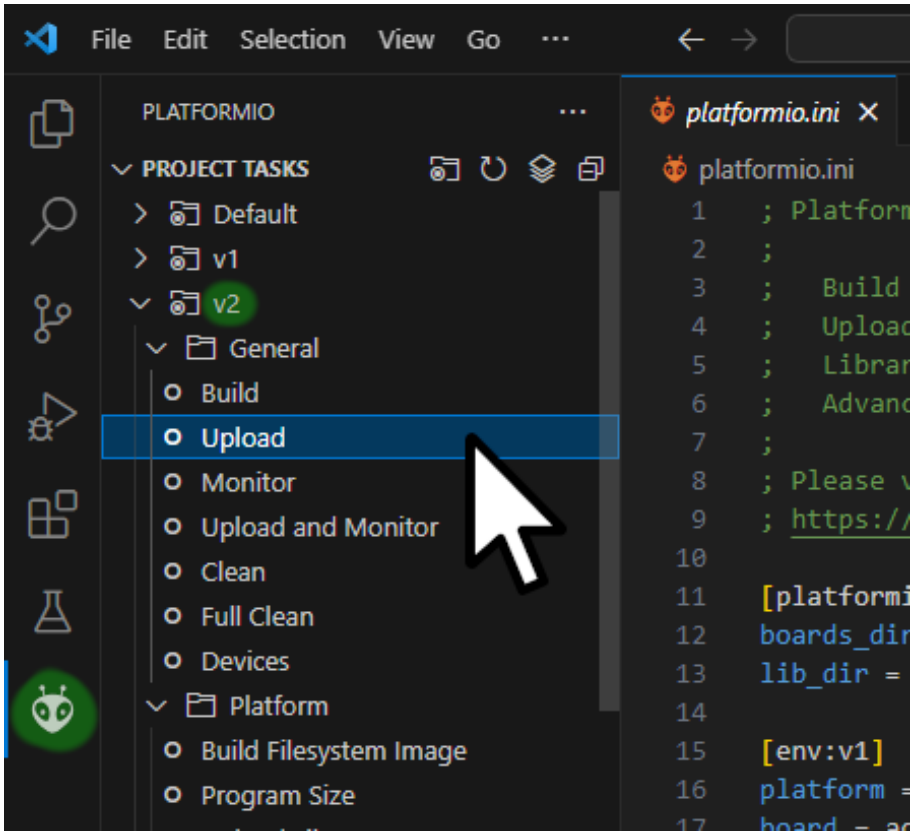
   Windows може вимагати встановлення драйверів для Лілки. Якщо вам буде запропоновано встановити драйвери, встановіть їх.

   Після цього вимкніть Лілку.

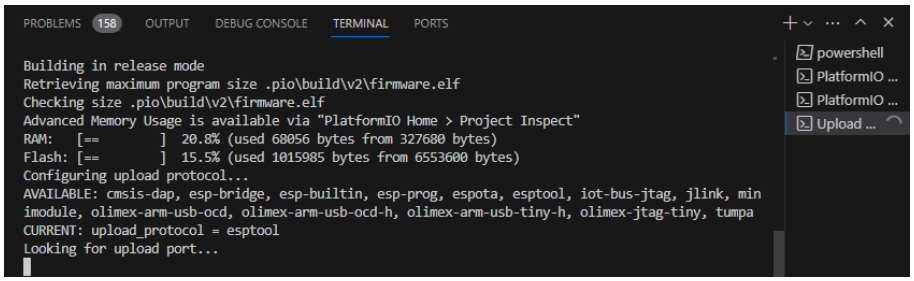
   Тепер вам потрібно натиснути та утримувати кнопку SELECT і увімкнути Лілку. Після цього можна відпустити кнопку SELECT. Це переведе Лілку в режим завантаження (bootloader mode): в цьому режимі ви можете завантажити в неї нову прошику.

2. Тепер перейдіть на вкладку PlatformIO та виберіть v2 в якості цільової платформи. Потім натисніть «Upload».

   Спершу PlatformIO виконає підготовчі дії, а потім розпочне завантаження нашої свіжозібраної прошивки в Лілку.



Початок завантаження виглядатиме ось так:



Тепер потрібно дочекатися завершення завантаження. Щойно завантаження завершиться, ви побачите повідомлення про успішне завершення завантаження:

TODO: додати скріншот, оскільки я це все робив під віртуалкою без Лілки під рукою. :)

3. Коли завантаження завершиться, натисніть кнопку RESET на Лілці та насолоджуйтесь новою прошивкою!

Порада: Тепер вам не потрібно щоразу вручну переводити Лілку в режим завантаження, щоб завантажити нову прошивку!

Крім прошивки Keira, ви щойно завантажили в Лілку ще й програму-завантажувач (bootloader). Тепер ви можете завантажувати нові прошивки в Лілку безпосередньо з PlatformIO.

Просто натисніть кнопку Upload в PlatformIO та вона автоматично переведе Лілку в режим завантаження та завантажить нову прошивку!

5.2.3 Вітаємо!

Вітаємо, ви успішно налаштували середовище розробки та завантажили Keira в Лілку!

Ви можете потицяти Лілку та переконатися, що вона працює і спробувати різні демо. Також ви можете перейти на вкладку Explorer (Ctrl + Shift + E), відкрити файл src/main.cpp та поглянути, як виглядає код прошивки, яку ми щойно зібрали і запустили на Лілці.

Також ви можете запускати готові прошивки прямо з SD-карти, а також писати власні програми на Lua та запускати їх з SD-карти. Детальніше - в наступній частині документації: Можливості.

Лілка - це відкрита платформа, тому ви можете змінювати код прошивки, додавати нові функції та вдосконалювати Лілку, або навіть створювати власні прошивки для неї. Але перш ніж приступати до цього, рекомендуємо ознайомитися з документацією бібліотеки lilka, яка спрощує роботу з Лілкою, за цим посиланням.

5.3 Карта microSD

Keira підтримує карти пам'яті microSD для запуску програм та зберігання даних.

Перш ніж використовувати карту пам'яті, вона повинна бути відформатована в файлову систему FAT32. Це можна зробити за допомогою програми форматування в операційній системі Windows або за допомогою команди mkfs.vfat в операційній системі GNU/Linux.

5.3.1 Форматування карти пам'яті в GNU/Linux

```
# Визначаємо пристрій, який відповідає карті пам'яті (наприклад, /dev/sdX)
lsblk
# Створюємо нову таблицю розділів MBR та один розділ FAT32, який займає всю
↪карту пам'яті
parted /dev/sdX --script -- mklabel msdos mkpart primary fat32 1MiB 100%
# Форматуємо розділ FAT32 з міткою LILKA
mkfs.vfat -F 32 -n LILKA /dev/sdX1
```

5.3.2 Форматування карти пам'яті в Windows

1. Підключіть карту пам'яті до комп'ютера.
2. Відкрийте провідник Windows та знайдіть карту пам'яті в списку пристроїв.
3. Клацніть правою кнопкою миші на карті пам'яті та виберіть «Форматувати».
4. Виберіть файлову систему FAT32 та натисніть кнопку «Почати».
5. Після завершення форматування виберіть «Закрити».

5.3.3 Наповнення карти пам'яті демонстраційними програмами

Перш ніж використовувати карту пам'яті з Keira, ви можете записати на неї демонстраційні програми.

Вони знаходяться в директорії firmware/keira/sdcard в репозиторії проєкту на GitHub.

Підключіть карту пам'яті до комп'ютера та скопіюйте файли з директорії firmware/keira/sdcard на карту пам'яті. Після цього відключіть карту пам'яті від комп'ютера та вставте її в Лілку.

Коли Лілка запуститься, виберіть пункт «Браузер SD-карти» в головному меню і спробуйте запустити одну з Lua-програм! (Вони мають розширення .lua і синю іконку.

5.4 Написання програм на C++

Зміст

5.4.1 Що таке програма в Keira?

Keira написана на C++, і вона містить ряд вбудованих програм. Програма в Keira - це клас, який наслідує клас App та визначає метод App::run(). Цей метод викликається при запуску програми.

Всі програми є вбудованими, тобто вони мають бути частиною Keira. Це означає, що ви не можете додати свою програму до Keira, якщо не зміните код Keira і не перепрошиєте Лілку новим кодом.

Ви можете використовувати будь-які функції з бібліотеки Lilka.

Примітка: Чи можу я написати свою програму на C++ окремо від Keira, якось скомпілювати її і запустити в Keira з SD-карти?

Ні. Динамічне завантаження програм у вбудованих системах - це дуже складний процес. Keira наразі не підтримує цю функцію. Але це може змінитись в майбутньому.

Якщо ви хочете писати програми, не перепрошиваючи Лілку, ми радимо спробувати вам Lua: Написання програм на Lua.

5.4.2 Клас App

Для створення власної програми вам потрібно наслідувати клас App та визначити метод App::run(). Цей метод буде викликатися при запуску програми.

Ось перелік важливих методів та властивостей класу App:

class App

Клас, що представляє додаток для Кіри.

Додатки запускаються за допомогою синглтону AppManager.

Додаток має визначити принаймні метод run(), який буде викликатися в окремій задачі FreeRTOS.

При завершенні додатку, AppManager зупиняє задачу та видаляє об'єкт додатку.

Приклад запуску додатку:

```
#include <appmanager.h>
#include <myapp.h>

// ...

AppManager::getInstance()->runApp(new MyApp());
```

Subclassed by AbstractLuaRunnerApp, BallApp, CubeApp, DemoLines, DiskApp, EpilepsyApp, FTPServerApp, GPIOManagerApp, KeyboardApp, LauncherApp, LetrisApp, LilTrackerApp, MJSApp, ModPlayerApp, NesApp, PetPetApp, ScanI2CApp, StatusBarApp, TamagotchiApp, TransformApp, UserSPIApp, WeatherApp, WiFiConfigApp

Public Functions

void queueDraw()

Повідомити ОС, що додаток завершив малювання кадру.

Цей метод слід викликати після малювання кожного кадру.

Технічно, цей метод міняє місцями передній буфер (canvas) та задній буфер (backCanvas).

Якщо цей метод викликається в той час, коли ОС вже малює попередній кадр, то він призведе до нетривалого блокування додатку. Іншими словами, якщо ваш додаток малює кадри швидше, ніж ОС здатна їх відображати, то цей метод буде час від часу заповільнювати ваш додаток. Це не проблема, але варто про це пам'ятати.

Public Members

lilka::Canvas *canvas

Вказівник на передній буфер для малювання.

Додаток повинен використовувати цей буфер для малювання всієї графіки.

Після малювання, буфер потрібно відобразити на екрані за допомогою методу queueDraw().

Protected Functions

void setFlags(AppFlags flags)

Встановити прапорці додатку.

Наприклад, якщо додаток має відображатися на весь екран, то слід викликати setFlags(APP_FLAG_FULLSCREEN).

Параметри
flags –

void setStackSize(uint32_t stackSize)

Встановити розмір стеку задачі додатку.

За замовчуванням, розмір стеку задачі дорівнює 8192 байт. Проте деякі додатки можуть вимагати більший розмір стеку.

Private Functions

virtual void run() = 0

Основний код додатку.

Цей метод викликається в окремій задачі FreeRTOS.

Додаток завершується, коли цей метод завершується або робить return.

inline virtual void onSuspend()

Цей метод викликається операційною системою, коли вона збирається зупинити ваш додаток.

inline virtual void onResume()

Цей метод викликається операційною системою, коли вона збирається відновити роботу вашого додатку.

inline virtual void onStop()

Цей метод викликається операційною системою, коли вона збирається зупинити ваш додаток.

5.4.3 Приклад програми

Давайте створимо просту програму, яка буде малювати круг на екрані, який можна переміщувати за допомогою кнопок.

Для цього створіть два нові файли в директорії firmware/keira/src/apps:

Listing 1: myapp.h

```
#include <lilka.h>
#include "app.h"

class MyApp : public App {
public:
    MyApp();
private:
    void run() override;
};
```

Listing 2: myapp.cpp

```
#include "myapp.h"

MyApp::MyApp() : App("Моя програма") {
}

void MyApp::run() {
    int16_t x = canvas->width() / 2;
    int16_t y = canvas->height() / 2;
    while (true) {
        // читаємо стан кнопок
        lilka::State state = lilka::controller.getState();

        if (state.up.pressed) {
            y--;
        } else if (state.down.pressed) {
            y++;
        }
        if (state.left.pressed) {
            x--;
        } else if (state.right.pressed) {
            x++;
        }
        if (state.a.pressed) {
            // Завершуємо програму
            return;
        }

        // заповнюємо екран чорним кольором
        canvas->fillScreen(canvas->color565(0, 0, 0));
        // малюємо білий круг
```

(continues on next page)

(continued from previous page)

```
        canvas->fillCircle(x, y, 10, canvas->color565(255, 255, 255));

        // повідомляємо Keira, що буфер змінився і його потрібно перемалювати
        queueDraw();
    }
}
```

Давайте розберемося з кодом.

1. Ми створили клас MyApp, який наслідує клас App.

   App містить в собі віртуальний метод run, який викликається при запуску програми.

   Також App автоматично створює об'єкт canvas, який представляє собою буфер для малювання. Ви повинні малювати саме на ньому, а не на екрані. Детальніше про це - згодом.

2. Весь код нашої програми знаходиться в методі run. Він автоматично викликається при запуску програми.

   Програма виконується в циклі while (true). Це означає, що вона буде виконуватися постійно, поки ви не викличете return.

3. Ми читаємо стан кнопок за допомогою lilka::controller.getState(). Це повертає об'єкт lilka::State, який містить в собі стан кожної кнопки.

   Наприклад, state.up.pressed - це true, якщо кнопка up натиснута.

4. Ми щоразу заповнюємо екран чорним кольором, малюємо білий круг, а потім викликаємо queueDraw().

   Цей метод повідомляє Keira, що буфер змінився і його потрібно перемалювати.

   Примітка: Чому ми не малюємо безпосередньо на екрані, і чому щоразу заповнюємо його чорним кольором? І що таке* queueDraw()?

   Це все пов'язано з тим, що Keira - це мультизадачна операційна система, і різні програми можуть намагатись одночасно малювати щось на екрані.

   Щоб уникнути конфліктів, Keira використовує подвійну буферизацію. Це означає, що кожна програма має два власні буфери: один («передній») для малювання, а інший («задній») - для відображення на екрані.

   - canvas - це передній буфер. Саме на ньому ваша програма малює все, що ви хочете побачити на екрані.
   - backCanvas - це задній буфер. Вам не потрібно ним керувати.

   Коли ви викликаєте метод queueDraw(), Keira міняє місцями передній і задній буфери і через деякий час починає малювати задній буфер на екрані в фоновому режимі. Таким чином ваша програма ніколи не малює безпосередньо на екрані: це робить Keira, а конкретніше - клас AppManager.

   canvas завжди вказує на передній буфер, тому ви повинні малювати саме на ньому. Але оскільки ці буфери постійно міняються місцями, ваша про-

грама не повинна робити жодних припущень про те, що було намальовано в попередній ітерації.

Тому після кожного виклику queueDraw() кожна програма повинна знову малювати все, що ви хочете побачити на екрані, оскільки canvas буде містити «сміття», а не те, що ви малювали в попередній ітерації, і завжди відставатиме на одну ітерацію від того, що відображається на екрані.

Це дає можливість не лише здійснювати конкурентне малювання на екрані з декількох програм, але й використовувати для цього обидва ядра процесора: одне ядро виконує вашу програму, а інше - перемальовує екран. Це збільшує FPS (кількість кадрів в секунду) і дозволяє досягнути максимальної утилізації процесора.

Майте на увазі, що виклик queueDraw() може заблокувати вашу програму на деякий час. Це ставатиметься в ситуаціях, коли Кіра ще не завершила малювати на екрані попередній буфер, а ви вже викликаєте queueDraw() знову. Це - не проблема, але варто про це пам'ятати.

В середньому, малювання займає близько 1/30 секунди. Це означає, що ви можете викликати queueDraw() близько 30 разів в секунду без блокування вашої програми.

5.4.4 Реєстрація програми в меню програм

Основна програма, що запускається при завантаженні Кіри, називається Launcher. Вона відповідає за відображення меню програм, налаштувань, інформації, а також запуск програм.

Щоб програма з'явилася в меню програм, вам потрібно зареєструвати її в одному з меню Launcher. Найпростіший спосіб - це додати вашу програму в меню додатків. Для цього знайдіть наступний код у файлі launcher.cpp та додайте вашу програму в список програм:

Listing 3: launcher.cpp

```
#include "myapp.h"  // <--- підключаємо вашу програму

// ...

ITEM_LIST app_items = {
    ITEM_SUBMENU(
        "Демо",
        {
            ITEM_APP("Лінії", DemoLines),
            ITEM_APP("Диск", DiskApp),
            ITEM_APP("Перетворення", TransformApp),
            ITEM_APP("М'ячик", BallApp),
            ITEM_APP("Куб", CubeApp),
            ITEM_APP("Епілепсія", EpilepsyApp),
        }
    ),
    ITEM_SUBMENU(
```

(continues on next page)

(continued from previous page)

```
        "Тести",
        {
            ITEM_APP("Клавіатура", KeyboardApp),
            ITEM_APP("Тест SPI", UserSPIApp),
            ITEM_APP("I2C-сканер", ScanI2CApp),
        },
    ),
    ITEM_APP("Летріс", LetrisApp),
    ITEM_APP("Тамагочі", TamagotchiApp),
    ITEM_APP("Моя програма", MyApp),  // <--- додаємо вашу програму
};
```

Після цього перепрошийте Лілку, і ваша програма з'явиться в меню програм.

5.5 Написання програм на Lua

Keira має вбудовану віртуальну машину мови програмування Lua. Це дозволяє писати та виконувати програми на Lua прямо з SD-картки, без необхідності компіляції чи перепрошивання Лілки.

> Попередження: Якщо ви не знайомі з мовою Lua, рекомендуємо прочитати туторіал з Lua перед тим, як продовжувати: https://www.lua.org/pil/1.html

5.5.1 Приклад програми

Оскільки Lua сама по собі - це універсальна мова програмування, вона не має вбудованих функцій для роботи з дисплеєм, звуком тощо. Тому для роботи з цими пристроями на Лілці в Keira доступні спеціальні вбудовані модулі, які надають доступ до функцій пристроїв.

Модулі завантажуються автоматично - не потрібно писати жодних require(...). Ось приклад простої програми на Lua, яка виводить текст «Hello, world!» на екран:

```
function lilka::update()
    if controller.get_state().a.just_pressed then
        -- Завершуємо програму при натисканні кнопки "A"
        util.exit()
    end
end

```

(continues on next page)

(continued from previous page)

```
function lilka::draw()
    -- Заповнюємо екран чорним кольором:
    display.fill_screen(display.color565(0, 0, 0))

    -- Виводимо текст "Hello, world!" на екран:
    display.set_cursor(0, 32)
    display.print("Hello, world!")
end
```

Ви можете зберегти цей код у файл з розширенням .lua на SD-картці, а потім виконати його, обравши його в браузері SD-картки.

Повний перелік доступних модулів та їх функцій можна знайти в розділі Lua API.

5.5.2 init, update, draw

В іграх важливо, щоб гра була плавною та виконувалася з певною стабільною кількістю кадрів на секунду.

Імовірно, ви знайомі з функцією delay(...) з фреймворка Arduino, яка затримує виконання програми на певну кількість мілісекунд. Це зручно для простих програм, що не використовують дисплей, але для ігор це не підходить, оскільки такі функції заблокують виконання програми на певну кількість мілісекунд, незалежно від швидкості виконання програми, а також не дозволяють вам взаємодіяти з грою під час затримки.

Для цього існує можливість визначити функції lilka.init(), lilka.update() і lilka.draw(). Якщо при запуску вашої програми Keira знайде цю функції, вона буде викликати їх автоматично.

- lilka.init() викликається один раз при запуску програми.
- lilka.update() викликається 30 разів на секунду, тому ви можете використовувати його для оновлення стану гри та обробки введення користувача.
- lilka.draw() викликається після lilka.update() та використовується для малювання графіки на екрані.

> Попередження: Програма на Lua буде виконуватись доти, доки не буде викликано функцію util.exit(). Інакше для виходу вам доведеться перезавантажити Лілку.

> Попередження: Не варто використовувати util.sleep() всередині ваших функцій lilka.update() та lilka.draw(), оскільки це призведе до заповільнення виконання програми.
>
> Намагайтесь писати код, який не блокує виконання програми, а використовує функцію lilka.update() для оновлення стану гри та обробки введення користувача.

lilka.update() також отримує необов'язковий аргумент delta, який вказує (в секундах), скільки часу пройшло з початку її попереднього виклику. Це дозволяє вам робити рухи та анімації, які будуть відбуватись з однаковою швидкістю незалежно від швидкості виконання програми.

За ідеальних обставин, delta буде дорівнювати 1/30, або приблизно 0.0333 секунди, але якщо код гри дуже складний і його виконання займає більше часу, ніж 1/30 мекунди, то значення delta буде більшим. Ваша програма може використовувати delta для того, щоб, наприклад, рухати об'єкти на екрані залежно від часу, а не від кількості кадрів на секунду.

Ці три функції повинні бути визначені у головному файлі програми, наприклад:

```
local ball_x
local ball_y

-- Завантажуємо зображення м'яча, яке знаходиться в корені SD-картки:
local ball = resources.load_image("ball.bmp", display.color565(255, 255, 255))

function lilka.init()
    -- Ця функція викликається один раз при запуску програми.
    -- Цей код можна було б виконати в глобальному контексті (поза цією функцією), як
↪ми це зробили з "ball",
    -- але ініціалізація гри буде очевиднішою, якщо вона відбувається тут.
    ball_x = display.width / 2
    ball_y = display.height / 2
end

function lilka.update(delta)
    local dir_x = 0
    local dir_y = 0

    -- Обробляємо введення користувача:
    local state = controller.get_state()
    if state.up.pressed then
        dir_y = -1
    elseif state.down.pressed then
        dir_y = 1
    end
    if state.left.pressed then
        dir_x = -1
    elseif state.right.pressed then
        dir_x = 1
    end
    if state.a.pressed then
        -- Вихід з програми:
        util.exit()
    end

    -- Переміщуємо м'яч зі швидкістю 50 пікселів на секунду
    ball_x = ball_x + dir_x * 50 * delta
    ball_y = ball_y + dir_y * 50 * delta
end

function lilka.draw()
    -- Малюємо графіку:
    display.fill_screen(display.color565(0, 0, 0))
```

(continues on next page)

(continued from previous page)

```
    display.draw_image(ball, ball_x, ball_y)

    -- Після виконання цієї функції Лілка автоматично відобразить все, що ми_
↪намалювали на екрані.
    -- Не потрібно викликати display.queue_draw() чи щось подібне.
end

-- Інші функції:
-- ...
```

Цей код створить просту програму, в якій ви можете керувати м'ячем за допомогою стрілок на контролері. Кожен кадр гри м'яч переміщується на певну відстань залежно від натиснутих кнопок, а потім малюється на екрані.

Завдяки аргументу delta м'яч завжди рухатиметься з однаковою швидкістю незалежно від того, як швидко виконується програма - чи це 30 кадрів на секунду, чи 10, чи 1000.

> Попередження: Ваш код всередині функції lilka.draw() не повинен робити жодних припущень щодо того, що вже було намальовано раніше. Кожен кадр гри потрібно малювати повністю.
>
> Це потрібно тому, що Keira використовує подвійну буферизацію екрану, тобто ваша гра малює на передньому буфері, а в цей час Keira відображає задній буфер на екрані.
>
> Тому радимо починати кожен кадр гри з виклику display.fill_screen(display.color565(0, 0, 0)), щоб очистити передній буфер перед малюванням нового кадру, оскільки він може бути забруднений попереднім кадром.

5.5.3 Швидке тестування програм

Звісно, ви можете зберегти вашу програму на SD-картці, а потім вибрати її в браузері SD-картки, але це може бути доволі незручно, особливо якщо ви працюєте над програмою, яка вимагає багато ітерацій. Щоразу, коли ви зберігаєте програму на SD-картці, ви повинні виймати її з Лілки, вставляти в комп'ютер, зберігати файл, виймати з комп'ютера, вставляти в Лілку, вибирати файл в браузері SD-картки, запускати програму, перевіряти, виправляти помилки, зберігати знову, О НІ! Це - нестерпно і в нас немає часу на це!

Саме тому Keira має функцію, яка називається Live Lua. Вона дозволяє вам запускати Lua-код на Лілці через USB-кабель прямо з вашого комп'ютера, без необхідності зберігати його на SD-картці.

> Попередження: Якщо ваша Lua-програма завантажує додаткові ресурси з SD-картки - наприклад, зображення, звуки тощо - ви спершу повинні вручну скопіювати ці ресурси на SD-картку, оскільки Live Lua не підтримує завантаження файлів на SD-картку через USB-кабель. Вона надсилає на Лілку лише код Lua.

> Попередження: Live Lua підтримує запуск програм, які складаються лише з одного файлу. Якщо ваша програма складається з декількох файлів, які завантажуються через require(), вам потрібно спершу скопіювати всі модулі на SD-картку, а потім використовувати Live Lua для запуску головного файлу програми.

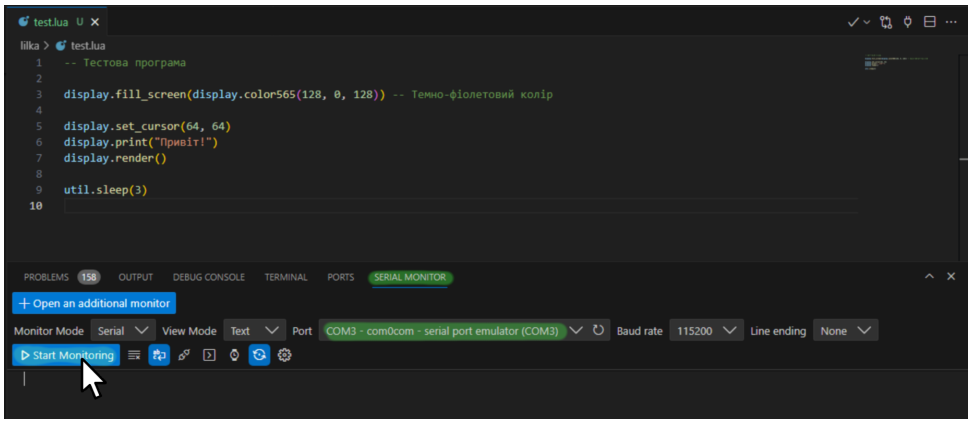
Щоб використовувати Live Lua, вам потрібно встановити розширення для Visual Studio Code, яке називається Serial Monitor:

Після цього підключіть Лілку до комп'ютера за допомогою USB-кабеля та перейдіть в меню «Розробка» -> «Live Lua». Лілка перейде в режим «Live Lua» та очікуватиме код Lua через USB-порт.

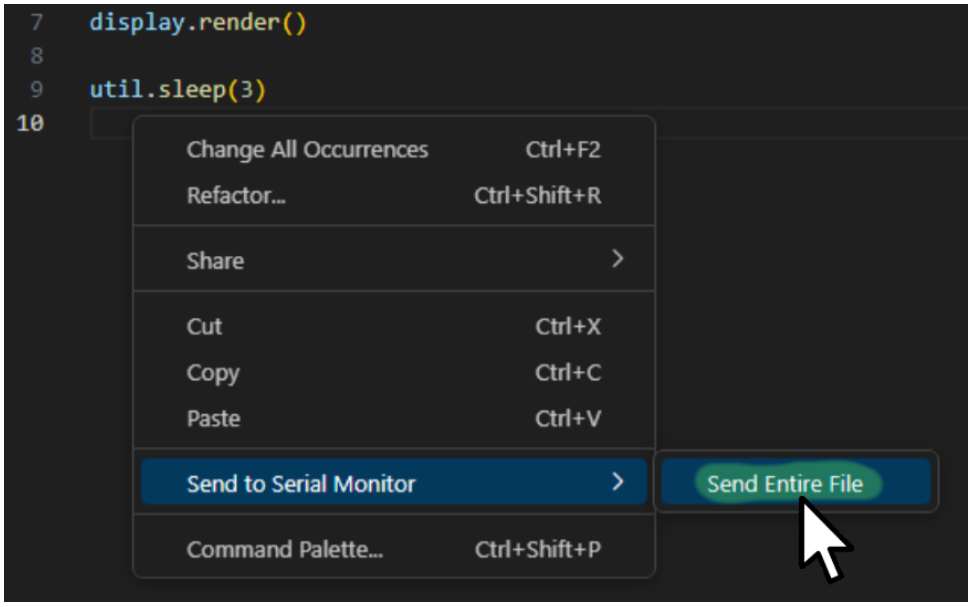
Далі створіть новий файл з розширенням .lua - наприклад, test.lua. Напишіть в ньому якийсь код:

```
display.fill_screen(display.color565(255, 0, 0))
display.set_cursor(50, 50)
display.print('Привіт, Лілка!')
display.queue_draw() -- Потрібно викликати цю функцію, щоб відобразити зміни на екрані, оскільки цей код знаходиться поза функцією lilka.draw
util.sleep(1)
```

Тепер перейдіть на вкладку «Serial Monitor» у VS Code, оберіть COM-порт, на якому підключена Лілка, та натисніть «Start Monitoring»:



Після цього натисніть правою кнопкою будь-де в коді на вкладці з вашим файлом test.lua та оберіть «Send to Serial Monitor» -> «Send Entire File»:



Вжух - і код миттєво запуститься на Лілці! Якщо ви зробите зміни в коді, ви можете просто натиснути «Send to Serial Monitor» -> «Send Entire File» знову, і новий код запуститься на Лілці. Слід лише переконатись, що попередня версія вашої програми завершилася, перш ніж ви надсилати нову версію.

5.5.4 Інтерактивна консоль Lua (REPL)

REPL означає «Read-Eval-Print Loop». Це - інтерактивне середовище, яка дозволяє вам друкувати код на клавіатурі та виконувати його на Лілці прямо з вашого комп'ютера через USB-кабель, по одному рядку коду за раз.

Це зручно для тестування функцій, вивчення API, відлагодження тощо.

Для запуску Lua REPL вам потрібно відкрити на Лілці меню «Розробка» -> «Lua REPL». Лілка перейде в режим «Lua REPL» та очікуватиме команди через USB-порт.

Для цього відкрийте вкладку «Serial Monitor» у VS Code, оберіть COM-порт, на якому підключена Лілка, натисніть «Start Monitoring» і почніть вводити код Lua. Після кожної команди тисніть Enter. Вона виконається на Лілці, і ви побачите результат в консолі Visual Studio Code.

Спробуйте ввести, наприклад, ось такий код:

```
print(3 + 4)
```

Після введення цього рядка ви маєте побачити в консолі результат виразу 3 + 4, тобто число 7.

Тепер спробуйте ввести цей код:

```
display.fill_screen(display.color565(255, 0, 0))
```

Після введення другого рядка та натиску на клавішу Enter, екран Лілки має забарвитися червоним кольором.

Щоб вийти з режиму REPL, натисніть на Лілці кнопку A.

5.6 Lua API

Документація всіх функцій, які доступні при написанні програм для Лілки мовою Lua.

5.6.1 lilka: Спеціальні функції

Цей модуль описує функції, які ви можете визначити в своєму коді, щоб вони викликались автоматично в певних ситуаціях.

Детальніше про їх поведінку можна прочитати в секції про написання ігор на Lua.

lilka.init()

Ця функція автоматично викликається один раз при запуску програми.

Ви повинні визначити її в своєму коді, якщо ви хочете використовувати її для ініціалізації вашої програми.

lilka.update(delta)

Parameters
delta (number) – Час, який пройшов з останнього кадру.

Ця функція автоматично викликається кожен кадр. Тут виконується ігрова логіка. В параметрі delta передається час, який пройшов з останнього кадру.

Ви повинні визначити її в своєму коді, якщо ви хочете використовувати її для оновлення вашої програми.

lilka.draw()

Ця функція автоматично викликається після lilka.update. Тут відбувається відображення графіки.

Ви повинні визначити її в своєму коді, якщо ви хочете використовувати її для малювання вашої програми.

class lilka

show_fps: boolean

Показувати частоту кадрів на екрані, якщо встановлено в true. Наприклад: lilka.show_fps = true

5.6.2 display: Дисплей

Основні функції для роботи з дисплеєм.

Приклад:

```
display.set_cursor(0, 0)
display.print("Hello,", "world!", 69, 420, "nice")

local color = display.color565(255, 0, 0)
display.draw_line(0, 0, 100, 100, color)

local face = resources.load_image("face.bmp", display.color565(0, 0, 0))
```

(continues on next page)

(continued from previous page)

```
display.draw_image(face, 50, 80)

display.fill_rect(0, 0, 100, 100, color)

display.queue_draw()
```

class display

width: integer

ширина дисплею в пікселях

height: integer

висота дисплею в пікселях

static color565(r, g, b)

Повертає 16-бітне значення кольору

Наприклад, display.color565(255, 0, 0) поверне 63488, що відповідає червоному кольору. Це потрібно, оскільки дисплей Лілки працює з 16-бітними кольорами (5-6-5), а не з 24-бітними (8-8-8).

Всі функції, що приймають аргумент color, очікують, що він буде саме 16-бітним числом.

Parameters

- r (integer) – червоний (0-255)
- g (integer) – зелений (0-255)
- b (integer) – синій (0-255)

Returns

16-бітне значення кольору (5-6-5)

Return type

integer

Usage:

```
local color = display.color565(255, 0, 0)
display.draw_line(0, 0, 100, 100, color)
```

static set_cursor(x, y)

Встановлює позицію курсора.

Позиція курсора використовується для виведення тексту на екран.

Parameters

- x (integer) – координата x
- y (integer) – координата y

static set_font(font)

Встановлює шрифт для виведення тексту.

Шрифт, який буде використовуватися для виведення тексту.

Доступні шрифти:

- "4x6": https://github.com/olikraus/u8g2/wiki/fntgrpx11#4x6
- "5x7": https://github.com/olikraus/u8g2/wiki/fntgrpx11#5x7
- "5x8": https://github.com/olikraus/u8g2/wiki/fntgrpx11#5x8
- "6x12": https://github.com/olikraus/u8g2/wiki/fntgrpx11#6x12
- "6x13": https://github.com/olikraus/u8g2/wiki/fntgrpx11#6x13
- "7x13": https://github.com/olikraus/u8g2/wiki/fntgrpx11#7x13
- "8x13": https://github.com/olikraus/u8g2/wiki/fntgrpx11#8x13
- "9x15": https://github.com/olikraus/u8g2/wiki/fntgrpx11#9x15
- "10x20": https://github.com/olikraus/u8g2/wiki/fntgrpx11#10x20

Parameters
font (str)

Usage:

```
display.set_font("6x13")
display.set_cursor(8, 32)
display.print("Привіт,")
display.set_cursor(8, 64)
display.set_font("10x20")
display.print("Лілка!")
```

static set_text_size(size)

Встановлює масштаб тексту.

Якщо цей параметр дорівнює 1, текст виводиться в масштабі 1:1. Якщо 2, то кожен піксель тексту буде займати 2x2 пікселі на екрані, і так далі.

Parameters
size (integer) – масштаб тексту

static print(...)

Виводить текст на екран.

Parameters
vararg (any) – значення для виведення

Usage:

```
display.set_cursor(8, 32)
display.print("Hello,", "world!", 69, 420, "nice")
```

static fill_screen(color)

Заповнює екран вказаним кольором.

Parameters
color (integer) – Колір (5-6-5)

Usage:

```
display.fill_screen(display.color565(255, 0, 0)) -- заповнює екран червоним кольором
```

static draw_pixel(x, y, color)

Малює піксель на екрані.

Parameters

- x (integer) – координата x
- y (integer) – координата y
- color (integer) – Колір (5-6-5)

static draw_line(x1, y1, x2, y2, color)

Малює лінію на екрані.

Parameters

- x1 (integer) – координата x початку лінії
- y1 (integer) – координата y початку лінії
- x2 (integer) – координата x кінця лінії
- y2 (integer) – координата y кінця лінії
- color (integer) – Колір (5-6-5)

Usage:

```
display.draw_line(0, 0, 100, 100, display.color565(255, 0, 0)) -- малює червону лінію від (0, 0) до (100, 100)
```

static draw_rect(x, y, w, h, color)

Малює контур прямокутника з координатами (x, y), розмірами (w, h) та кольором color.

Parameters

- x (integer) – координата x початку прямокутника
- y (integer) – координата y початку прямокутника
- w (integer) – ширина прямокутника
- h (integer) – висота прямокутника
- color (integer) – Колір (5-6-5)

static fill_rect(x, y, w, h, color)

Малює заповнений прямокутник з координатами (x, y), розмірами (w, h) та кольором color.

Parameters

- x (integer) – координата x початку прямокутника
- y (integer) – координата y початку прямокутника
- w (integer) – ширина прямокутника

- h (integer) – висота прямокутника
- color (integer) – Колір (5-6-5)

static draw_circle(x, y, r, color)

Малює контур кола з центром в точці (x, y), радіусом r та кольором color.

Parameters

- x (integer) – координата x центру кола
- y (integer) – координата y центру кола
- r (integer) – радіус кола
- color (integer) – Колір (5-6-5)

static fill_circle(x, y, r, color)

Малює заповнене коло з центром в точці (x, y), радіусом r та кольором color.

Parameters

- x (integer) – координата x центру кола
- y (integer) – координата y центру кола
- r (integer) – радіус кола
- color (integer) – Колір (5-6-5)

static draw_triangle(x1, y1, x2, y2, x3, y3, color)

Малює контур трикутника з вершинами в точках (x1, y1), (x2, y2), (x3, y3) та кольором color.

Parameters

- x1 (integer) – координата x першої точки
- y1 (integer) – координата y першої точки
- x2 (integer) – координата x другої точки
- y2 (integer) – координата y другої точки
- x3 (integer) – координата x третьої точки
- y3 (integer) – координата y третьої точки
- color (integer) – Колір (5-6-5)

static fill_triangle(x1, y1, x2, y2, x3, y3, color)

Малює заповнений трикутник з вершинами в точках (x1, y1), (x2, y2), (x3, y3) та кольором color.

Parameters

- x1 (integer) – координата x першої точки
- y1 (integer) – координата y першої точки
- x2 (integer) – координата x другої точки
- y2 (integer) – координата y другої точки
- x3 (integer) – координата x третьої точки

- y3 (integer) – координата y третьої точки
- color (integer) – Колір (5-6-5)

static draw_elipse(x, y, rx, ry, color)

Малює контур еліпса з центром в точці (x, y), радіусами rx та ry та кольором color.

Parameters

- x (integer) – координата x центру еліпса
- y (integer) – координата y центру еліпса
- rx (integer) – радіус еліпса по осі x
- ry (integer) – радіус еліпса по осі y
- color (integer) – Колір (5-6-5)

static fill_elipse(x, y, rx, ry, color)

Малює заповнений еліпс з центром в точці (x, y), радіусами rx та ry та кольором color.

Parameters

- x (integer) – координата x центру еліпса
- y (integer) – координата y центру еліпса
- rx (integer) – радіус еліпса по осі x
- ry (integer) – радіус еліпса по осі y
- color (integer) – Колір (5-6-5)

static draw_arc(x, y, r1, r2, start_angle, end_angle, color)

Малює контур дуги з центром в точці (x, y), зовнішнім радіусом r1, внутрішнім радіусом r2, початковим кутом start_angle, кінцевим кутом end_angle та кольором color.

Parameters

- x (integer) – координата x центру дуги
- y (integer) – координата y центру дуги
- r1 (integer) – зовнішній радіус дуги
- r2 (integer) – внутрішній радіус дуги
- start_angle (integer) – початковий кут дуги (в градусах)
- end_angle (integer) – кінцевий кут дуги (в градусах)
- color (integer) – Колір (5-6-5)

static fill_arc(x, y, r1, r2, start_angle, end_angle, color)

Малює заповнену дугу з центром в точці (x, y), зовнішнім радіусом r1, внутрішнім радіусом r2, початковим кутом start_angle, кінцевим кутом end_angle та кольором color.

Parameters

- x (integer) – координата x центру дуги
- y (integer) – координата y центру дуги
- r1 (integer) – зовнішній радіус дуги
- r2 (integer) – внутрішній радіус дуги
- start_angle (integer) – початковий кут дуги (в градусах)
- end_angle (integer) – кінцевий кут дуги (в градусах)
- color (integer) – Колір (5-6-5)

static draw_image(image, x, y)

Малює зображення на екрані.

Parameters

- image (table) – ідентифікатор зображення
- x (integer) – координата x лівого верхнього кута зображення
- y (integer) – координата y лівого верхнього кута зображення

Usage:

```
local image = resource.load_image("face.bmp", display.color565(0, 0, 0))
display.draw_image(image, 50, 80) -- малює зображення в точці (50, 80)
```

static draw_image_transformed(image, x, y, transform)

Малює зображення на екрані з перетворенням.

Parameters

- image (table) – ідентифікатор зображення
- x (integer) – координата x лівого верхнього кута зображення
- y (integer) – координата y лівого верхнього кута зображення
- transform (Transform) – перетворення

Usage:

```
local image = resource.load_image("face.bmp", display.color565(0, 0, 0))
local transform = transforms.new()
transform = transform:rotate(45)
display.draw_image(image, 50, 80, transform) -- малює зображення в точці (50,
↪80) з перетвореннями
```

static queue_draw()

Примусово оновлює вміст екрану.

> Попередження: Ми не радимо використовувати цю функцію. Вона викликається автоматично після кожного виконання lilka.draw(). Вам слід писати весь код малювання всередині своєї функції lilka.draw(), а цю функцію використовувати лише для тестування простих програм.

Return type
nil

5.6.3 transform - Перетворення зображень

Функції для обертання та масштабування зображень.

Приклад:

```
local image = resource.load_image("face.bmp", display.color565(0, 0, 0))
local transform = transforms.new()
transform = transform:scale(1.5, 0.5)
transform = transform:rotate(45)
display.draw_image(image, 50, 80, transform) -- малює зображення в точці (50, 80) з
↪перетвореннями
```

class transforms

static new()
Створити нове афінне перетворення.

Return type
Transform

class Transform

scale(x, y)
Масштабувати перетворення по обох осях та повернути нове перетворення.

Parameters
- x (number) – масштаб по осі X
- y (number) – масштаб по осі Y

Return type
Transform

rotate(angle)
Обернути перетворення на певний кут та повернути нове перетворення.

Parameters
angle (number) – кут повороту (в градусах)

Return type
Transform

multiply(other)
Помножити перетворення на інше перетворення та повернути результат, не змінюючи поточне перетворення.

Parameters
other (Transform) – інше перетворення

Return type
Transform

inverse()

Отримати перетворення, яке є оберненим до поточного.

Return type
Transform

vtransform(x, y)

Застосувати перетворення до точки та повернути нові координати.

Parameters

- x (number) – координата X
- y (number) – координата Y

Return type
number or number

Usage:

```
local transform = transforms.new():scale(2, 2):rotate(45)
local x, y = transform:transform(10, 10)
print(x, y)
```

get()

Отримати матрицю перетворення.

Returns
двовимірна таблиця 2x2

Return type
table

set(matrix)

Встановити матрицю перетворення.

> Попередження: Ми не радимо вам використовувати цей метод, якщо ви не знаєте, що робите. Краще використовуйте методи scale та rotate.

Parameters
matrix (table) – двовимірна таблиця 2x2

Usage:

```
local transform = transforms.new()
transform:set({{1, 0}, {0, 1}})
```

5.6.4 controller - Введення

Функції для роботи з введенням.

Приклад:

```
display.set_cursor(32, 32)

while true do
    local state = controller.get_stage()

    if state.a.just_pressed then
        print("A щойно натиснуто!")
    elseif state.a.just_released then
        print("A щойно відпущено!")
    end
end
```

class controller

static get_state()

Повертає таблицю зі станом контролера.

Ця таблиця містить наступні поля:

- up: стан кнопки «вгору»
- down: стан кнопки «вниз»
- left: стан кнопки «вліво»
- right: стан кнопки «вправо»
- a: стан кнопки «A»
- b: стан кнопки «B»
- c: стан кнопки «C»
- d: стан кнопки «D»
- select: стан кнопки «SELECT»
- start: стан кнопки «START»

Кожна поле має наступні підполя:

- pressed: true, якщо кнопка натиснута
- just_pressed: true, якщо кнопка щойно натиснута вперше з моменту попереднього виклику controller.get_state
- just_released: true, якщо кнопка щойно відпущена вперше з моменту попереднього виклику controller.get_state

Return type
table

Usage:

```
display.set_cursor(0, 32)

while true do
    local state = controller.get_state()

    if state.a.just_pressed then
        display.print("[A] щойно натиснуто!")
    elseif state.a.just_released then
        display.print("[A] щойно відпущено!")
    end
end
```

5.6.5 resources - Ресурси

Функції для роботи з ресурсами (зображеннями, звуками тощо).

Приклад:

```
local face = resources.load_image("face.bmp", display.color565(0, 0, 0))
display.draw_image(face, 50, 80)
```

class resources

static load_image(filename, transparent_color?, pivotX?, pivotY?)

Завантажує BMP-зображення і повертає таблицю з ідентифікатором зображення (а також з його розмірами), яку можна використовувати для малювання зображення на екрані.

Ця таблиця містить наступні поля:

- width: ширина зображення
- height: висота зображення
- pointer: внутрішній ідентифікатор зображення

Parameters

- filename (str) – шлях до файлу зображення .BMP (відносно місця знаходження скрипта, що виконується)
- transparent_color? (integer) – колір, який буде використаний для прозорості (5-6-5). Якщо цей параметр не вказаний, зображення буде виводитись без прозорості
- pivotX? (integer) – X-координата центру зображення (за замовчуванням це середина зображення)
- pivotY? (integer) – Y-координата центру зображення (за замовчуванням це середина зображення)

Return type
table

Usage:

```lua
local face = resources.load_image("face.bmp", display.color565(0, 0, 0))
print(face.width, face.height) -- Виведе розміри зображення
display.draw_image(face, 50, 80) -- Виведе зображення на екран у позицію (50, 80)
```

static rotate_image(image, angle, blank_color)

Повертає зображення на певну кількість градусів за годинниковою стрілкою навколо його центру.

Parameters

- image (table) – ідентифікатор зображення
- angle (integer) – кут (в градусах)
- blank_color (integer) – колір для пікселів, які залишаться незаповненими

static flip_image_x(image)

Відображає зображення горизонтально. Зручно використовувати для платформерів, де герой може рухатись вліво та вправо.

Parameters
image (table) – ідентифікатор зображення

static flip_image_y(image)

Відображає зображення вертикально.

Parameters
image (table) – ідентифікатор зображення

static read_file(filename)

Читає вміст файлу і повертає його як текст.

Parameters
filename (str) – шлях до файлу (відносно місця знаходження скрипта, що виконується)

Return type
str

Usage:

```lua
local content = resources.read_file("file.txt")
print(content) -- Виведе вміст файлу
```

static write_file(filename, content)

Записує текст у файл.

Parameters

- filename (str) – шлях до файлу (відносно місця знаходження скрипта, що виконується)
- content (str) – текст, який буде записаний у файл

Usage:

```
resources.write_file("file.txt", "Hello, world!")
```

5.6.6 math - Арифметичні функції

Цей модуль містить функції для виконання різних арифметичних та тригонометричних операцій.

class math

pi: number

Число

e: number

Число e

tau: number

Число (2)

static random(a?, b?)

Повертає випадкове число.

- Якщо заданий лише один аргумент, повертає випадкове ЦІЛЕ число в діапазоні [0;a] (включно).
- Якщо задані обидва аргументи, повертає випадкове ЦІЛЕ число в діапазоні [a;b] (включно).
- Якщо не задані аргументи, повертає випадкове ДРОБОВЕ число в діапазоні [0;1] (включно).

Parameters

- a? (integer) – початок діапазону (включно)
- b? (integer) – кінець діапазону (включно)

Return type

number

Usage:

```
local r = math.random(10, 20)
print(r) -- Виведе випадкове число від 10 до 20 (включно). Можливо, це буде
↪13? А можливо, русня - не люди?
```

static clamp(x, min, max)

Обмежує число x в діапазоні між min та max (включно).

Parameters

- x (number) – число, яке потрібно обмежити
- min (number) – мінімальне значення
- max (number) – максимальне значення

Return type
number

Usage:

```
print(math.clamp(8.1, 10, 20)) -- Виведе 10
print(math.clamp(15.2, 10, 20)) -- Виведе 15.2
print(math.clamp(23.3, 10, 20)) -- Виведе 20
```

static lerp(min, max, t)

Лінійна інтерполяція.

Повертає значення, яке лінійно інтерполюється між min та max з коефіцієнтом t.

Parameters

- min (number) – мінімальне значення
- max (number) – максимальне значення
- t (number) – коефіцієнт інтерполяції (від 0 до 1)

Return type
number

Usage:

```
print(math.lerp(0, 100, 0.5)) -- Виведе 50
print(math.lerp(0, 100, 0.25)) -- Виведе 25
print(math.lerp(0, 100, 0.7125)) -- Виведе 71.25
```

static map(x, in_min, in_max, out_min, out_max)

Перетворення значення з одного діапазону в інший.

Повертає число x, перетворене з діапазону [in_min;in_max] в діапазон [out_min;out_max].

Parameters

- x (number) – число, яке потрібно перетворити
- in_min (number) – початок вхідного діапазону
- in_max (number) – кінець вхідного діапазону
- out_min (number) – початок вихідного діапазону
- out_max (number) – кінець вихідного діапазону

Return type
number

Usage:

```
print(math.map(50, 0, 100, 0, 1)) -- Виведе 0.5
print(math.map(25, 0, 100, 0, 1)) -- Виведе 0.25
print(math.map(71.25, 0, 100, 0, 1)) -- Виведе 0.7125
```

static abs(x)

Повертає модуль числа x.

Parameters
x (number) – число

Return type
number

static sign(x)

Повертає знак числа x: -1, якщо число від'ємне, 0, якщо число дорівнює 0, 1, якщо число додатнє.

Parameters
x (number) – число

Return type
number

Usage:

```
print(math.sign(-5)) -- Виведе -1
print(math.sign(5)) -- Виведе 1
print(math.sign(0)) -- Виведе 0
```

static sqrt(x)

Повертає квадратний корінь числа x.

Parameters
x (number) – число

Return type
number

static pow(base, exp)

Повертає число x в степені exp.

Parameters

- base (number) – число
- exp (number) – степінь

Return type
number

Usage:

```
print(math.pow(2, 3)) -- Виведе 8
print(math.pow(64, 0.5)) -- Виведе 8
```

static min(values)

Повертає мінімальне значення з таблиці.

Parameters
values (table) – таблиця чисел

Return type
number

Usage:

```
print(math.min({1.1, 2.2, 3.3, 4.4, 5.5})) -- Виведе 1.1
print(math.min({5, 4, 3, 2, 1})) -- Виведе 1
print(math.min({-5, -4, -3, -2, -1})) -- Виведе -5
```

static max(values)

Повертає максимальне значення з таблиці.

Parameters
values (table) – таблиця чисел

Return type
number

Usage:

```
print(math.max({1.1, 2.2, 3.3, 4.4, 5.5})) -- Виведе 5.5
print(math.max({5, 4, 3, 2, 1})) -- Виведе 5
print(math.max({-5, -4, -3, -2, -1})) -- Виведе -1
```

static sum(values)

Повертає суму всіх чисел з таблиці.

Parameters
values (table) – таблиця чисел

Return type
number

Usage:

```
print(math.sum({1, 2, 3, 4, 5})) -- Виведе 15
print(math.sum({1.1, 2.2, 3.3, 4.4, 5.5})) -- Виведе 16.5
print(math.sum({-5, -4, -3, -2, -1})) -- Виведе -15
```

static avg(values)

Повертає середнє значення всіх чисел з таблиці.

Parameters
values (table) – таблиця чисел

Return type
number

Usage:

```
print(math.avg({1, 2, 3, 4, 5})) -- Виведе 3
print(math.avg({1.1, 2.2, 3.3, 4.4})) -- Виведе 2.75
print(math.avg({-5, -4, -3, -2})) -- Виведе -3.5
```

static floor(x)

Округлює число x вниз.

Повертає найбільше ціле число, яке менше або дорівнює x.

Parameters
x (number) – число

Return type
integer

Usage:

```
print(math.floor(1.1)) -- Виведе 1
print(math.floor(1.9)) -- Виведе 1
print(math.floor(-1.1)) -- Виведе -2
print(math.floor(-1.9)) -- Виведе -2
```

static ceil(x)

Округлює число x вгору.

Повертає найменше ціле число, яке більше або дорівнює x.

Parameters
x (number) – число

Return type
integer

Usage:

```
print(math.ceil(1.1)) -- Виведе 2
print(math.ceil(1.9)) -- Виведе 2
print(math.ceil(-1.1)) -- Виведе -1
print(math.ceil(-1.9)) -- Виведе -1
```

static round(x)

Округлює число x до найближчого цілого.

Parameters
x (number) – число

Return type
integer

Usage:

```
print(math.round(1.1)) -- Виведе 1
print(math.round(1.9)) -- Виведе 2
print(math.round(-1.1)) -- Виведе -1
print(math.round(-1.9)) -- Виведе -2
```

static sin(x)

Повертає значення синуса кута x (в радіанах).

Parameters
x (number) – кут (в радіанах)

Return type
number

Usage:

```
print(math.sin(0)) -- Виведе 0
print(math.sin(math.pi / 2)) -- Виведе 1
```

static cos(x)

Повертає значення косинуса кута x (в радіанах).

Parameters
x (number) – кут (в радіанах)

Return type
number

Usage:

```
print(math.cos(0)) -- Виведе 1
print(math.cos(math.pi)) -- Виведе -1
```

static tan(x)

Повертає значення тангенса кута x (в радіанах).

Parameters
x (number) – кут (в радіанах)

Return type
number

Usage:

```
print(math.tan(0)) -- Виведе 0
print(math.tan(math.pi / 4)) -- Виведе 1
print(math.tan(math.pi / 2)) -- Виведе якесь дуже велике число, оскільки тангенс не визначений для кутів, що дорівнюють /2.
                             -- Чому? Подумайте самі: трикутник з кутом 180 градусів - це взагалі трикутник?
```

static asin(x)

Повертає значення арксинуса числа x.

Parameters
x (number) – число

Return type
number

static acos(x)

Повертає значення арккосинуса числа x.

Parameters
x (number) – число

Return type
number

static atan(x)

Повертає значення арктангенса числа x.

Parameters
x (number) – число

Return type
number

static atan2(y, x)

Повертає значення арктангенса числа y/x.

Цей метод дуже зручно використовувати для обчислення кута між віссю x та точкою (x, y). А взагалі, якщо ви розумієте, що таке арктангенс, то, можливо, вам варто спробувати писати код на C? :)

Parameters

- y (number) – число y
- x (number) – число x

static log(x, base?)

Повертає значення логарифму числа x.

Parameters

- x (number) – число
- base? (number) – основа логарифму (за замовчуванням - число e)

Return type
number

static deg(x)

Перетворює кут x з радіанів в градуси.

Parameters
x (number) – кут (в радіанах)

Return type
number

static rad(x)

Перетворює кут x з градусів в радіани.

Parameters
x (number) – кут (в градусах)

Return type
number

static norm(x, y)

Нормалізує вектор (x, y) до одиничної довжини.

Parameters

- x (number) – координата x
- y (number) – координата y

Return type
number or number

static len(x, y)

Повертає довжину вектора (x, y).

Parameters

- x (number) – координата x
- y (number) – координата y

Return type
number

Usage:

```
print(math.len(3, 4)) -- Виведе 5
print(math.len(1, 1)) -- Виведе 1.4142135623731
```

static dist(x1, y1, x2, y2)

Повертає відстань між точками (x1;y1) та (x2;y2).

Parameters

- x1 (number) – координата x першої точки
- y1 (number) – координата y першої точки
- x2 (number) – координата x другої точки
- y2 (number) – координата y другої точки

Return type
number

Usage:

```
print(math.dist(0, 0, 3, 4) -- Виведе 5
print(math.dist(0, 0, 1, 1) -- Виведе 1.4142135623731
```

static rotate(x, y, angle)

Повертає вектор (x, y), обернутий на кут angle за годинниковою стрілкою (якщо уявити, що вісь Y вказує вниз, як це прийнято в комп'ютерній графіці).

Parameters

- x (number) – координата x
- y (number) – координата y
- angle (number) – кут (в градусах)

Return type
number or number

Usage:

```
local x, y = math.rotate(1, 0, 45)
print(x, y) -- Виведе 0, 1
```

5.6.7 geometry - Геометричні функції

Цей модуль містить функції, які будуть корисні для створення ігор, де потрібно працювати з геометричними об'єктами та колізіями.

class geometry

Геометричні функції

static intersect_lines(ax, ay, bx, by, cx, cy, dx, dy)

Повертає true, якщо відрізки AB та CD перетинаються.

Parameters

- ax (number) – координата x першої точки відрізка AB
- ay (number) – координата y першої точки відрізка AB
- bx (number) – координата x другої точки відрізка AB
- by (number) – координата y другої точки відрізка AB
- cx (number) – координата x першої точки відрізка CD
- cy (number) – координата y першої точки відрізка CD
- dx (number) – координата x другої точки відрізка CD
- dy (number) – координата y другої точки відрізка CD

Return type
boolean

static intersect_aabb(ax, ay, aw, ah, bx, by, bw, bh)

Повертає true, якщо прямокутник (ax, ay, aw, ah) перетинається з прямокутником (bx, by, bw, bh).

Parameters

- ax (number) – координата x верхнього лівого кута першого прямокутника
- ay (number) – координата y верхнього лівого кута першого прямокутника
- aw (number) – ширина першого прямокутника
- ah (number) – висота першого прямокутника
- bx (number) – координата x верхнього лівого кута другого прямокутника
- by (number) – координата y верхнього лівого кута другого прямокутника
- bw (number) – ширина другого прямокутника
- bh (number) – висота другого прямокутника

Return type
boolean

5.6.8 gpio - Керування GPIO-пінами

Функції для роботи з GPIO-пінами роз'єму розширення.

Приклад:

```
-- Ця програма блимає світлодіодом, під'єднаним до піна 12 через резистор.

led_pin = 12

-- Встановлюємо пін 12 в режим виводу
gpio.mode(led_pin, gpio.OUTPUT)
while true do
    -- Вмикаємо світлодіод
    gpio.write(led_pin, gpio.HIGH)
    util.sleep(0.5)
    -- Вимикаємо світлодіод
    gpio.write(led_pin, gpio.LOW)
    util.sleep(0.5)
    if controller.get_state().a.just_pressed then
        -- Кнопку A щойно натиснули, зупиняємо цикл
        break
    end
end
```

class gpio

LOW: integer
: логічне значення 0

HIGH: integer
: логічне значення 1

INPUT: integer
: режим введення

OUTPUT: integer
: режим виведення

INPUT_PULLUP: integer
: режим введення з підтяжкою вгору

INPUT_PULLDOWN: integer
: режим введення з підтяжкою вниз

static set_mode(pin, mode)
: Налаштовує режим GPIO-піна.

  Parameters

  - pin (integer) – номер піна
  - mode (integer) – режим піна, може бути gpio.INPUT, gpio.OUTPUT, gpio.INPUT_PULLUP або gpio.INPUT_PULLDOWN

static write(pin, value)

Записує цифрове зрачення в GPIO-пін.

Parameters

- pin (integer) – номер піна
- value (integer) – цифрове значення, може бути gpio.LOW або gpio.HIGH (можна використовувати 0 або 1)

static read(pin)

Читає цифрове значення з GPIO-піна. Воно може бути gpio.LOW або gpio.HIGH (можна використовувати 0 або 1 для порівняння).

Parameters
pin (integer) – номер піна

Return type
integer

static analog_read(pin)

Читає аналогове значення з GPIO-піна. Воно може бути в діапазоні від 0 до 4095 (включно).

Майте на увазі, що ця функція працює тільки з пінами, які підтримують аналогове введення. Детальніше про це - на сторінці Роз'єм розширення.

Parameters
pin (integer) – номер піна

Return type
integer

5.6.9 util - Утиліти

Різні корисні функції.

class util

static sleep(sec)

Затримує виконання скрипта на вказану кількість секунд.

Parameters
sec (number) – кількість секунд, на яку потрібно затримати виконання програми

Usage:

```
display.set_cursor(0, 32)
display.print("Зачекайте півсекунди...")
util.sleep(0.5) -- Затримує виконання програми на півсекунди.
display.print("Готово!")
```

static exit()

Завершує виконання програми.

Return type
nil

Usage:

```
function lilka._update(delta)
    state = controller.get_state()
    if state.a.pressed then
        util.exit() -- Завершує виконання програми.
    end
    print('Staying alive!')
end
```

5.6.10 buzzer - П'єзо-динамік

Функції для роботи з п'єзо-динаміком.

Примітка: Ці функції не блокують виконання програми: всі звуки та мелодії відтворюються в фоновому режимі.

---

class buzzer

static play(frequency, size?)

Відтворює звук заданої частоти.

Якщо передати другий аргумент, звук буде відтворено впродовж цього часу (в мілісекундах).

Parameters

- frequency (number) – частота тону
- size? (number) – тривалість звуку (в мілісекундах)

static play_melody(melody, tempo)

Відтворює мелодію.

Мелодія - це масив з пар частота-розмірність.

Наприклад, ноту з частотою 523 Гц (нота «до» п'ятої октави) і тривалістю 1/4 можна представити як {523, 4}.

Від'ємна тривалість означає ноту з крапкою, наприклад:

- -1 - ціла нота з крапкою (1 + 1/2)
- -2 - половина з крапкою (1/2 + 1/4)
- -4 - чверть з крапкою (1/4 + 1/8)

. . . і так далі.

Якщо частота дорівнює 0, то вона інтерпретується як пауза.

Parameters

- melody (table) – мелодія (масив пар частота-розмірність)

- tempo (number) – темп мелодії (кількість ударів на хвилину)

Usage:

```
-- Мелодія "до-ре-мі-фа-соль" п'ятої октави
local melody = {
    {523, 2},
    {587, 4},
    {659, 2},
    {698, 4},
    {784, 2},
}
buzzer.play_melody(melody, 60)  -- Відтворює мелодію з темпом 60 ударів на хвилину
-- Також можна використовувати глобальний модуль notes, де є константи для нот від B0 до D#8:
local melody2 = {
    {notes.C4, 4},  -- "до" четвертої октави
    {notes.FS4, 2}, -- "фа-дієз" четвертої октави
    {notes.D5, 1},  -- "ре" п'ятої октави
    {0, 2},  -- Пауза
}
buzzer.play_melody(melody2, 60)
```

static stop()

Зупиняє відтворення всіх звуків.

Return type
nil

5.6.11 sdcard - Робота з SD-картою

Функції для роботи читання/запису файлів на SD-карті.

Приклад:

```
local file = sdcard.open("file.txt", "w")
file:write("Hello, world!")
```

class sdcard

static ls(path)

Повернути таблицю зі списком файлів та директорій за вказаним шляхом.

Parameters
path (str) – шлях до директорії (відносно кореня SD-картки)

Return type
table

Usage:

```lua
local entries = sdcard.ls("/folder")
for i = 0, #entries do
    print(entries[i])
end
```

static remove(path)

Видалити файл або директорію за вказаним шляхом.

Parameters
path (str) – шлях до файлу або директорії (відносно кореня SD-картки)

Usage:

```lua
sdcard.remove("/folder/file.txt")
```

static rename(old_path, new_path)

Перейменувати файл або директорію.

Parameters

- old_path (str) – старий шлях до файлу або директорії (відносно кореня SD-картки)
- new_path (str) – новий шлях до файлу або директорії (відносно кореня SD-картки)

Usage:

```lua
sdcard.rename("/folder/file.txt", "/folder/file2.txt")
```

static open(path, mode)

Відкрити файл за вказаним шляхом.

Parameters

- path (str) – шлях до файлу (відносно кореня SD-картки)
- mode (str) – режим відкриття файлу (див. функцію fopen у документації ANSI C)

Return type
File

Usage:

```lua
local file = sdcard.open("/file.txt", "a+") -- Відкриває файл для додавання тексту
file:write("Hello, world!\n") -- Дописує текст в кінець файлу
```

class File

size()

Повернути розмір файлу.

Return type
integer

seek(pos)

Перемістити вказівник файлу на певну позицію.

Parameters
pos (integer) – позиція в файлі

read(count)

Прочитати з файлу.

Parameters
count (integer) – максимальна кількість байт, які потрібно прочитати

Return type
str

write(content)

Записати у файл.

Parameters
content (str) – дані, які потрібно записати

Usage:

```lua
local file = sdcard.open("/file.txt", "w") -- Відкриває файл для запису
file:write("Hello, world!\n") -- Записує текст у файл
file:write("Привіт, світ!\n") -- Дописує текст у файл
```

5.6.12 state - Збереження стану програми

class state

Сховище для зберігання даних програми.

Якщо вам потрібно зберегти дані між запусками програми (наприклад, налаштування або стан гри), ви можете використовувати цей об'єкт.

Цей об'єкт автоматично зберігається на диск при завершенні програми, і відновлюється при наступному запуску.

Всі дані зберігаються в тій самій директорії, що і програма, в файлі з розширенням .state. Наприклад, якщо файл вашої програми називається mygame.lua, то дані будуть збережені в файлі mygame.state.

> Попередження: Дозволено зберігати тільки прості типи даних: string, number, boolean та nil. Вкладені таблиці, функції та інші типи не підтримуються і будуть проігноровані.

Usage:

```lua
-- Ця програма при кожному запуску збільшує лічильник на 1 та виводить його значення в консоль.

state = state or {} -- якщо це перший запуск, то state буде не визначений, тому ми
```

(continues on next page)

(continued from previous page)

```
↪ініціалізуємо його пустою таблицею
if state.counter == nil then
    state.counter = 0
end
state.counter = state.counter + 1
display.fill_screen(display.color565(64, 0, 64))
display.set_cursor(0, 64)
display.print('Лічильник запусків\nпрограми: ', state.counter)
display.queue_draw()
util.sleep(0.5)
```

5.6.13 wifi - Робота з WiFi-мережами

Функції для роботи з WiFi-мережами.

Приклад:

```
local networks = wifi.scan()
for i = 0, #networks do
    print(networks[i])
    print('Сила сигналу: ', wifi.get_rssi(i))
    print('Тип шифрування: ', wifi.get_encryption_type(i))
end
```

class wifi

static connect(ssid, password)

Під'єднатись до мережі Wi-Fi.

Parameters

- ssid (str) – ім'я мережі
- password (str) – пароль мережі

static get_status()

Отримати статус мережі Wi-Fi.

Returns
статус мережі (див. WiFi.status() у документації Arduino ESP32)

Return type
integer

static scan()

Повернути таблицю з назвами доступних мереж Wi-Fi.

Return type
table

Usage:

```lua
local networks = wifi.scan()
for i = 0, #networks do
    print(networks[i])
    print('Сила сигналу: ', wifi.get_rssi(i))
    print('Тип шифрування: ', wifi.get_encryption_type(i))
end
```

static get_rssi(index)

Отримати силу сигналу відповідної мережі Wi-Fi.

Parameters
index (integer) – індекс мережі (порядковий номер у списку, отриманому функцією wifi.scan())

Return type
integer

static get_encryption_type(index)

Отримати тип шифрування відповідної мережі Wi-Fi.

Parameters
index (integer) – індекс мережі (порядковий номер у списку, отриманому функцією wifi.scan())

Return type
integer

static get_mac()

Отримати MAC-адресу пристрою.

Return type
str

static get_local_ip()

Отримати локальну IP-адресу пристрою.

Return type
str

static set_config(ip, gateway, subnet, dns1, dns2)

Налаштувати параметри мережі Wi-Fi для статичної IP-адреси.

(Див. функцію WiFi.config() у документації Arduino ESP32)

Parameters

- ip (str) – IP-адреса
- gateway (str) – IP-адреса шлюзу
- subnet (str) – маска підмережі
- dns1 (str) – IP-адреса DNS-сервера 1
- dns2 (str) – IP-адреса DNS-сервера 2

Розділ 6

Бібліотека lilka

Проєкт «Лілка» має однойменну бібліотеку lilka, яка спрощує роботу з Лілкою та дозволяє вам швидко створювати для Лілки власні прошивки мовою програмування C++.

Ця бібліотека доступна в реєстрі бібліотек PlatformIO.

Звісно ж, ви можете писати код для Лілки і без бібліотеки lilka - ніхто не змушує вас використовувати її. Весь код, який ви побачите в цій документації, можна написати самостійно, без використання lilka.

Крім того, ви без проблем можете використовувати для роботи з Лілкою лише функції Arduino, ESP-IDF та різні сторонні бібліотеки.

Проте бібліолека lilka допоможе зробити ваші програми простішими та більш «portable» (переносними), а також зменшить кількість помилок при розробці.

Також lilka спрощує ініціалізацію дисплею, кнопок, батареї, звуку, SD-карти та інших компонентів, що входять до складу Лілки. Для цього потрібно викликати лише одну функцію - lilka::begin().

Тому, незалежно від рівня вашої кваліфікації, ми радимо використовувати бібліолеку lilka для роботи з Лілкою.

void lilka::begin()

Ініціалізація Лілки

Ініціалізує всі підсистеми Лілки - дисплей, кнопки, файлову систему, SD-карту, батарею, п'єзо-динамік і т.д.

Рекомендується викликати цю функцію один раз на початку програми в вашій функції setup().

```
1 #include <lilka.h>
2
3 void setup() {
4     lilka::begin();
```

(continues on next page)

(continued from previous page)

```
    // Все залізо готове до роботи!
}

void loop() {
    // Заповнити екран чорним кольором
    lilka::display.fillScreen(lilka::colors::Black);

    while (1) {
        // Отримати стан кнопок
        lilka::State state = lilka::controller.getState();

        if (state.a.justPressed) { // Якщо щойно була натиснута кнопка "A"...
            // Розпочати відтворення звуку на частоті 440 Гц
            lilka::buzzer.play(lilka::NOTE_A4);
            // Заповнити екран червоним кольором
            lilka::display.fillScreen(lilka::colors::Red);
        } else if (state.a.justReleased) { // Якщо кнопка "A" щойно була відпущена...
            // Зупинити відтворення звуку
            lilka::buzzer.stop();
            // Заповнити екран зеленим кольором
            lilka::display.fillScreen(lilka::colors::Green);
        }
    }
}
```

Примітка: Порада для початківців: ви могли помітити, що всі функції та об'єкти бібліотеки lilka викликаються через ::. Це означає, що вони належать до простору назв lilka.

Такий синтаксис дозволяє нам уникнути плутанини з іншими бібліотеками (а також з вашим кодом), які можуть містити функції чи змінні з такими ж назвами.

Якщо не хочете щоразу писати lilka:: перед кожною функцією та об'єктом бібліотеки lilka, ви можете використати директиву using namespace:

```
#include <lilka.h>

using namespace lilka; // Тепер не потрібно щоразу писати "lilka::"

void setup() {
    begin();
    // Все залізо готове до роботи!
}

void loop() {
    // Заповнити екран чорним кольором
    display.fillScreen(colors::Black);

    while (1) {
        // Отримати стан кнопок
```

(continues on next page)

(continued from previous page)

```
        State state = controller.getState();

        if (state.a.justPressed) { // Якщо щойно була натиснута кнопка "A"...
            // Розпочати відтворення звуку на частоті 440 Гц
            buzzer.play(NOTE_A4);
            // Заповнити екран червоним кольором
            display.fillScreen(colors::Red);
        } else if (state.a.justReleased) { // Якщо кнопка "A" щойно була відпущена...
            // Зупинити відтворення звуку
            buzzer.stop();
            // Заповнити екран зеленим кольором
            display.fillScreen(colors::Green);
        }
    }
}
```

6.1 Audio: Звук (I2S)

class Audio

Клас для ініціалізації аудіо.

Цей клас лише встановлює піни для I2S і відтворює тестовий звук.

Для роботи з аудіо використовуйте клас I2S напряму: https://espressif-docs.readthedocs-hosted.com/projects/arduino-esp32/en/latest/api/i2s.html#sample-code

Public Functions

void begin()

Налаштоувує піни для I2S і відтворює тестовий звук.

> Попередження: Цей метод викликається автоматично при виклику lilka::begin().

void initPins()

Налаштувує піни для I2S. Цей метод варто викликати перед викликом i2s_driver_install().

6.2 Battery: Батарея

Battery lilka::battery

Екземпляр класу Battery, який можна використовувати для вимірювання рівня заряду акумулятора. Вам не потрібно інстанціювати Battery вручну.

LILKA_DEFAULT_EMPTY_VOLTAGE

Номінальне значення напруги LiPo акумулятора, при якій вважається, що він порожній.

LILKA_DEFAULT_FULL_VOLTAGE

Номінальне значення напруги LiPo акумулятора, при якій вважається, що він повністю заряджений.

class Battery

Клас для вимірювання рівня заряду акумулятора. Використовується для вимірювання напруги на акумуляторі через дільник напруги. Вимірювання проводиться через ADC.

Приклад використання:

```
#include <lilka.h>

void setup() {
    lilka::begin();
}

void loop() {
    int batteryLevel = lilka::battery.readLevel();
    Serial.println("Заряд батареї: " + String(batteryLevel) + "%");
    delay(1000);
}
```

Примітка: Цей клас не потрібно створювати вручну, оскільки він вже створений за замовчуванням і доступний як lilka::battery.

Public Functions

Battery()

void begin()

Почати вимірювання рівня заряду акумулятора.

> Попередження: Цей метод викликається автоматично при виклику lilka::begin().

int readLevel()

Прочитати рівень заряду акумулятора.

Повертає
Рівень заряду акумулятора від 0 до 100. Якщо акумулятор відсутній, повертається -1.

uint16_t readRawValue()

Прочитати сире значення АЦП напруги акумулятора.

Дивись також:

readLevel

Повертає
Значення АЦП напруги акумулятора від 0 до 4095.

void setEmptyVoltage(float voltage)

Встановити напругу акумулятора, при якій він вважається порожнім. За замовчуванням використовується значення LILKA_DEFAULT_EMPTY_VOLTAGE.

void setFullVoltage(float voltage)

Встановити напругу акумулятора, при якій він вважається повним. За замовчуванням використовується значення LILKA_DEFAULT_FULL_VOLTAGE.

6.3 Board: Керування платою

Board lilka::board

Екземпляр класу Board, який можна використовувати для керування платою. Вам не потрібно інстанціювати Board вручну.

class Board

Клас для керування платою.

Ініціалізує роз'єм розширення та режим енергозбереження.

Приклад використання:

```
#include <lilka.h>

void setup() {
    lilka.begin();
}

void loop() {
    lilka.board.enablePowerSavingMode(); // Вимкнути дисплей та I2S-модуль
```

(continues on next page)

(continued from previous page)

```
    ESP.deepSleep(1000000); // Перейти в режим глибокого сну на 1 секунду
    lilka.board.disablePowerSavingMode(); // Увімкнути дисплей та I2S-модуль
    delay(1000);
}
```

Public Functions

void begin()

Налаштувати плату.

> Попередження: Цей метод викликається автоматично при виклику lilka::begin().

void enablePowerSavingMode()

Увімкнути режим енергозбереження.

Цей метод вимикає дисплей, підсвітку дисплея та I2S-модуль. Його варто викликати перед входом в режим сну або глибокого сну.

void disablePowerSavingMode()

Вимкнути режим енергозбереження.

Цей метод вмикає дисплей, підсвітку дисплея та I2S-модуль. Його варто викликати після виходу з режиму сну.

uint8_t getExtPinGPIO(uint8_t index)

Отримати номер GPIO, що відповідає пінові з роз'єму розширення за індексом.

Повертає номер піна роз'єму розширення за індексом. Нульовий індекс - це пін, що має квадратну форму.

Дивись також:

ExtPin

Параметри
: index – Індекс піна роз'єму розширення.

Повертає
: Номер GPIO, що відповідає даному піну з роз'єму розширення. Якщо цей пін - спеціальний (наприклад, земля або живлення), повертається відповідний код з переліку lilka::ExtPin.

enum lilka::ExtPin

Коди для спеціальних пінів роз'єму розширення.

Values:

enumerator INVALID

Недійсний пін.

enumerator GND

Земля.

enumerator VCC

Живлення.

6.4 Buzzer: П'єзо-динамік

Buzzer lilka::buzzer

Екземпляр класу Buzzer, який можна використовувати для відтворення монотонних звуків. Вам не потрібно інстанціювати Buzzer вручну.

struct Tone

Public Members

uint16_t frequency

Частота ноти (може бути значенням з Note)

int8_t size

Розмір ноти:

- 1 - ціла нота
- 2 - половина
- 4 - чверть
- 8 - одна восьма і т.д.

Від'ємне значення - це ноти з крапкою:

- -1 - ціла нота з крапкою (1 + 1/2)
- -2 - половина з крапкою (1/2 + 1/4)
- -4 - чверть з крапкою (1/4 + 1/8)
- -8 - одна восьма з крапкою (1/8 + 1/16) і т.д.

class Buzzer

Клас для роботи з п'єзо-динаміком. Використовується для відтворення монотонних звуків.

Всі методи цього класу є неблокуючими, тобто вони не чекають завершення відтворення звуку і не блокують виконання коду, що йде після них.

Щоб зупинити відтворення звуку, використовуйте метод stop().

Приклад використання:

```
#include <lilka.h>

void setup() {
    lilka::begin();
}

void loop() {
    lilka::buzzer.play(lilka::NOTE_A4); // Грати ноту "Ля"
    delay(500);
    lilka::buzzer.stop(); // Зупинити відтворення
    delay(1500);
}
```

Public Functions

Buzzer()

void begin()

Почати роботу з п'єзо-динаміком.

> Попередження: Цей метод викликається автоматично при виклику lilka::begin().

void play(uint16_t frequency)

Відтворити ноту з певною частотою.

void play(uint16_t frequency, uint32_t duration)

Відтворити ноту з певною частотою впродовж певного часу.

void playMelody(const Tone *melody, uint32_t length, uint32_t tempo = 120)

Відтворити мелодію.

void stop()

Зупинити відтворення всіх звуків.

void playDoom()

Відтворити мелодію з DOOM - E1M1, At Doom's Gate (Bobby Prince).

Public Static Functions

static void melodyTask(void *arg)

6.5 Controller: Кнопки

Controller lilka::controller

Екземпляр класу Controller, який можна використовувати для вимірювання стану кнопок. Вам не потрібно інстанціювати Controller вручну.

struct ButtonState

Містить стан кнопки, який був вимінярий в певний момент часу.

Public Members

bool pressed

true, якщо кнопка була в натиснутому стані в момент виклику lilka::controller.getState().

bool justPressed

true, якщо кнопка була вперше натиснута в момент виклику lilka::controller.getState() (до цього була відпущена).

bool justReleased

true, якщо кнопка була вперше відпущена в момент виклику lilka::controller.getState() (до цього була натиснута).

uint64_t time

uint64_t nextRepeatTime

uint32_t repeatRate

uint32_t repeatDelay

struct State

Містить стани всіх кнопок, які були виміряні в певний момент часу.

Public Members

ButtonState up

ButtonState down

ButtonState left

ButtonState right

ButtonState a

ButtonState b

ButtonState c

ButtonState d

ButtonState select

ButtonState start

ButtonState any

Спеціальний стан, який містить стани «будь-якої» кнопки.

class Controller

Клас для роботи з контролером.

Використовується для вимірювання стану кнопок.

Приклад використання:

```
#include <lilka.h>

void setup() {
    lilka::begin();
}

void loop() {
    while (1) {
        lilka::State state = lilka::controller.getState();
        if (state.up.justPressed) {
            Serial.println("Ви щойно натиснули кнопку 'Вгору'");
        } else if (state.up.justReleased) {
            Serial.println("Ви щойно відпустили кнопку 'Вгору'");
```

(continues on next page)

(continued from previous page)

```
    }
  }
}
```

Public Functions

Controller()

void begin()

Почати вимірювання стану кнопок.

> Попередження: Цей метод викликається автоматично при виклику lilka::begin().

State getState()

Прочитати стан кнопок та скинути прапорці justPressed та justReleased.

State peekState()

Прочитати стан кнопок, не скидаючи прапорців justPressed та justReleased.

void resetState()

void setGlobalHandler(void (*handler)(Button, bool))

Встановити глобальний обробник подій, який буде викликатися при натисненні або відпусканні будь-якої кнопки.

void setHandler(Button button, void (*handler)(bool))

Встановити обробник подій для певної кнопки, який буде викликатися при натисненні або відпусканні цієї кнопки.

void clearHandlers()

Видалити всі обробники подій.

void setAutoRepeat(Button button, uint32_t rate, uint32_t delay)

Налаштувати автоматичне повторення натискання кнопки.

Після виклику цього методу кнопка буде автоматично натискатися з певною затримкою та частотою.

Щоб вимкнути автоматичне повторення натискання кнопки, викличте цей метод з параметрами delay = 0 та rate = 0.

Параметри

- button – Кнопка, для якої налаштовується автоматичне повторення натискання.
- rate – Частота автоматичного повторення натискання (кількість натискань на секунду).
- delay – Затримка перед початком автоматичного повторення натискання (в мілісекундах).

```
// Натискання кнопки "Вгору" буде повторюватись з частотою
↪5 натискань на секунду після початкової затримки 500 мс:
lilka::controller.setAutoRepeat(lilka::Button::UP, 5, 500);
```

6.6 Display: Дисплей

Display lilka::display

Екземпляр класу Display, який можна використовувати для роботи з дисплеєм. Вам не потрібно інстанціювати Display вручну.

class Display : public Arduino_ST7789, public lilka::GFX<Display>

Клас для роботи з дисплеєм.

Дивись також:

GFX

Використовується для відображення графічних об'єктів.

Цей клас наслідує Arduino_GFX з бібліотеки Arduino_GFX_Library, а також клас GFX.

Детальніше про доступні методи можна дізнатися в документації бібліотеки Arduino_GFX_Library - https://github.com/moononournation/Arduino_GFX.

```
#include <lilka.h>

void setup() {
    lilka.begin();
}

void loop() {
  lilka::display.fillScreen(lilka::colors::Red); // Заповнити екран червоним кольором
  lilka::display.setCursor(32, 32);
  lilka::display.setTextColor(lilka::colors::Green); // Зелений текст
  lilka::display.print("Привіт, Лілка!");
}
```

Public Functions

void begin()

Почати роботу з дисплеєм.

> Попередження: Цей метод викликається автоматично при виклику lilka::begin().

void setSplash(const void *splash, uint32_t rleLength = 0)

Встановити зображення, яке буде відображатися при запуску.

За замовчуванням відображається вітальний екран Лілки.

Його потрібно викликати перед викликом lilka::begin() або не викликати взагалі.

Примітка: Якщо викликати цей метод, то вітальний екран буде відображатись навіть якщо LILKA_NO_SPLASH встановлено в true.

Параметри

- splash – Масив 16-бітних кольорів (5-6-5) з розміром 280*240 (або масив байтів, закодованих алгоритмом RLE, з довжиною rleLength).
- rleLength – Якщо використовується RLE-кодування, цей аргумент вказує довжину масиву splash. Зображення повинне бути згенероване за допомогою утиліти sdk/tools/image2code з прапорцем --rle.

uint16_t color565hsv(uint16_t hue, uint8_t sat, uint8_t val)

Перетворити HSV колір в 16-бітний формат.

Параметри

- hue – Тон (0-360).
- sat – Насиченість (0-100).
- val – Яскравість (0-100).

Повертає
16-бітний колір.

class Canvas : public Arduino_Canvas, public lilka::GFX<Canvas>

Клас для роботи з графічним буфером. Він наслідує клас Arduino_Canvas з бібліотеки Arduino_GFX_Library, а також клас GFX.

Дивись також:

GFX

При частому перемальовуванні екрану без використання буфера може спостерігатися мерехтіння. Наприклад, якщо використовувати метод fillScreen для очищення екрану перед кожним викликом print, то текст буде мерехтіти.

Щоб уникнути цього, можна використовувати графічний буфер. Цей клас дозволяє малювати графічні об'єкти на буфері, а потім відобразити його на екрані за допомогою методу lilka::display.drawCanvas. Фактично, цей клас і є графічним буфером.

Такий підхід дозволяє зменшити мерехтіння, але збільшує використання пам'яті. Він називається «буферизація», оскільки ми спершу малюємо на буфері, а тоді відображаємо буфер на екрані.

Цей клас, як і Display, є підкласом Arduino_GFX з бібліотеки Arduino_GFX_Library. Це означає, що майже всі методи, які доступні в Display, також доступні в Canvas.

```
#include <lilka.h>

void setup() {
    lilka.begin();
}

void loop() {
    lilka::Canvas canvas; // Створити новий Canvas зі стандартним розміром (розмір дисплею)
    int y = 100;
    while (1) {
        canvas.fillScreen(lilka::colors::Black); // Заповнити буфер чорним кольором
        canvas.setCursor(32, y);
        canvas.setTextColor(lilka::colors::Black); // Чорний текст
        canvas.print("Привіт, Лілка!");
        lilka::display.drawCanvas(&canvas); // Відобразити буфер на екрані - жодного мерехтіння!
        y++;
        if (y > 200) {
            y = 100;
        }
    }
}
```

Public Functions

Canvas()

Створити буфер зі стандартним розміром (який дорівнює розміру дисплею).

Canvas(uint16_t w, uint16_t h)

Створити буфер з заданими розмірами.

Параметри

- w – Ширина буфера.
- h – Висота буфера.

Canvas(uint16_t x, uint16_t y, uint16_t w, uint16_t h)

Створити буфер з заданими розмірами та позицією.

Параметри

- x – Координата X лівого верхнього кута буфера.
- y – Координата Y лівого верхнього кута буфера.
- w – Ширина буфера.
- h – Висота буфера.

template<typename T>

class GFX

Цей клас описує спільні методи для класів Display та Canvas, оскільки вони обидва є підкласами GFX.

Public Functions

uint16_t color565(uint8_t r, uint8_t g, uint8_t b)

Перетворити RGB колір в 16-бітний формат.

Оскільки дисплей підтримує лише 16-бітні кольори, цей метод дозволяє перетворити 24-бітний колір в 16-бітний.

Параметри

- r – Компонента R.
- g – Компонента G.
- b – Компонента B.

void setFont(const uint8_t *font)

Встановити шрифт.

Шрифт можна вибрати зі списку рекомендованих шрифтів:

- FONT_4x6
- FONT_5x7
- FONT_5x8
- FONT_6x12
- FONT_6x13
- FONT_7x13
- FONT_8x13
- FONT_8x13_MONO
- FONT_9x15
- FONT_10x20
- FONT_10x20_MONO

Також можна використати будь-який інший шрифт з бібліотеки U8g2: https://github.com/olikraus/u8g2/wiki/fntlistallplain

```
lilka::display.setFont(FONT_6x12);
lilka::display.setCursor(0, 32);
lilka::display.print("Привіт, Лілка!");
```

Параметри
font – Вказівник на шрифт.

void setCursor(int16_t x, int16_t y)

Встановити курсор.

Параметри

- x – Координата X.
- y – Координата Y.

void setTextSize(uint8_t size)

Встановити масштаб тексту.

Якщо цей параметр дорівнює 1, текст виводиться в масштабі 1:1. Якщо 2, то кожен піксель тексту буде займати 2x2 пікселі на екрані, і так далі.

Параметри
size – Масштаб.

void setTextColor(uint16_t color)

Встановити колір тексту.

void setTextColor(uint16_t color, uint16_t background)

Встановити колір тексту та фону.

void print(...)

Відобразити текст.

```
lilka::display.setCursor(0, 32);
lilka::display.setTextColor(lilka::colors::Black); // Чорний текст
lilka::display.print("Привіт, ");
lilka::display.print(String("Лілка!\n"));
lilka::display.print(42);
```

Дивись також:

setCursor, setTextColor, setTextSize, setFont

Параметри
... – Текст.

void fillScreen(uint16_t color)

Заповнити екран кольором.

void drawPixel(int16_t x, int16_t y, uint16_t color)

Встановити колір пікселя.

void drawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint16_t color)

Намалювати лінію.

void drawRect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color)

Намалювати прямокутник.

void fillRect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color)

Намалювати заповнений прямокутник.

void drawCircle(int16_t x, int16_t y, int16_t r, uint16_t color)

Намалювати коло.

void fillCircle(int16_t x, int16_t y, int16_t r, uint16_t color)

Намалювати заповнене коло.

void drawTriangle(int16_t x0, int16_t y0, int16_t x1, int16_t y1, int16_t x2, int16_t y2, uint16_t color)

Намалювати трикутник.

void fillTriangle(int16_t x0, int16_t y0, int16_t x1, int16_t y1, int16_t x2, int16_t y2, uint16_t color)

Намалювати заповнений трикутник.

void drawEllipse(int16_t x, int16_t y, int16_t rx, int16_t ry, uint16_t color)

Намалювати еліпс.

Параметри

- x – Координата X центру еліпса.
- y – Координата Y центру еліпса.
- rx – Радіус по X.
- ry – Радіус по Y.
- color – Колір.

void fillEllipse(int16_t x, int16_t y, int16_t rx, int16_t ry, uint16_t color)

Намалювати заповнений еліпс.

Дивись також:

drawEllipse

void drawArc(int16_t x, int16_t y, int16_t r1, int16_t r2, int16_t start, int16_t end, uint16_t color)

Намалювати дугу.

Параметри

- x – Координата X центру дуги.
- y – Координата Y центру дуги.
- r1 – Зовнішній радіус дуги.
- r2 – Внутрішній радіус дуги.
- start – Початковий кут (в градусах).
- end – Кінцевий кут (в градусах).
- color – Колір.

void fillArc(int16_t x, int16_t y, int16_t r1, int16_t r2, int16_t start, int16_t end, uint16_t color)

Намалювати заповнену дугу.

Дивись також:

drawArc

void draw16bitRGBBitmap(int16_t x, int16_t y, uint16_t *bitmap, int16_t w, int16_t h)

Намалювати зображення з масиву 16-бітних точок.

```
lilka::Image *image = lilka::resources.loadImage("image.bmp");
lilka::display.drawBitmap(0, 0, image->pixels, image->width, image->height);
```

Параметри

- x – Координата X лівого верхнього кута зображення.
- y – Координата Y лівого верхнього кута зображення.
- bitmap – Масив 16-бітних кольорів.
- w – Ширина зображення.
- h – Висота зображення.

void draw16bitRGBBitmap(int16_t x, int16_t y, const uint16_t bitmap[], int16_t w, int16_t h)

Дивись також:

draw16bitRGBBitmap

void draw16bitRGBBitmapWithTranColor(int16_t x, int16_t y, uint16_t *bitmap, uint16_t transparent_color, int16_t w, int16_t h)

Намалювати зображення з масиву 16-бітних точок і вказати колір, який буде вважатися прозорим.

```
// Завантажити зображення з файлу "image.bmp", використовуючи білий колір як прозорий.
lilka::Image *image = lilka::resources.loadImage("image.bmp", lilka::colors::White);
lilka::display.draw16bitRGBBitmapWithTranColor(
    0, 0, image->pixels, image->transparentColor, image->width, image->height
);
```

Параметри

- x – Координата X лівого верхнього кута зображення.
- y – Координата Y лівого верхнього кута зображення.
- bitmap – Масив 16-бітних кольорів.
- transparent_color – Колір, який буде вважатися прозорим.
- w – Ширина зображення.

- h – Висота зображення.

void drawCanvas(Canvas *canvas)

Відобразити буфер на екрані (див. lilka::Canvas).

void drawImage(Image *image, int16_t x, int16_t y)

Намалювати зображення.

```
lilka::Image *image = lilka::resources.loadImage("image.bmp");
if (!image) {
    Serial.println("Failed to load image");
    return;
}
lilka::display.drawImage(image, 32, 64);
// Звільнюємо пам'ять
delete image;
```

Параметри

- image – Вказівник на зображення (об’єкт класу lilka::Image).
- x – Координата X осі зображення.
- y – Координата Y осі зображення.

void drawImageTransformed(Image *image, int16_t x, int16_t y, Transform transform)

Намалювати зображення з афінними перетвореннями.

Дивись також:

lilka::Transform

Примітка: Зверніть увагу, що перетворення - це повільніше, ніж звичайне малювання зображення, оскільки обчислює координати пікселів «на льоту». Використовуйте його лише тоді, коли не можете заздалегідь створити обернені копії зображеня за допомогою методів lilka::Image::rotate, lilka::Image::flipX та lilka::Image::flipY.

Параметри

- image – Вказівник на зображення (об’єкт класу lilka::Image).
- x – Координата X осі зображення.
- y – Координата Y осі зображення.
- transform – Об’єкт класу lilka::Transform, який містить матрицю перетворення.

class Image

Зображення

Містить розміри, прозорий колір та пікселі зображення (в 16-бітному форматі, 5-6-5). Пікселі зберігаються в рядку зліва направо, зверху вниз.

Вісь зображення (pivot) - це точка, яка вказує на центр зображення. Це дозволяє вам встановити точку, відносно якої буде відображатися зображення, а також навколо якої буде відбуватися перетворення зображення.

Примітка: Основна відмінність Image від поняття «bitmap» погялає в тому, що Image містить масив пікселів, розміри зображення і прозорий колір, в той час як «bitmap» - це просто масив пікселів.

Public Functions

Image(uint32_t width, uint32_t height, int32_t transparentColor = -1, int16_t pivotX = 0, int16_t pivotY = 0)

Створити зображення з заданими розмірами та прозорим кольором.

Якщо transparentColor встановлено в -1, то прозорість відсутня.

Параметри

- width – Ширина зображення.
- height – Висота зображення.
- transparentColor – 16-бітний колір (5-6-5), який буде вважатися прозорим. За замовчуванням -1 (прозорість відсутня).
- pivotX – Координата X центральної осі зображення. За замовчуванням 0.
- pivotY – Координата Y центральної осі зображення. За замовчуванням 0.

void rotate(int16_t angle, Image *dest, int32_t blankColor)

Обернути зображення на заданий кут (в градусах) і записати результат в dest.

Цей метод, а також методи flipX та flipY, зручно використовувати для створення обернених та віддзеркалених копій зображення, якщо ви заздалегідь знаєте, які варіанти зображення вам знадобляться.

Замість них можна використовувати клас lilka::Transform та його методи, які дозволяють виконувати та комбінувати складніші перетворення «на льоту», але такі перетворення є повільнішими.

```
lilka::Image *image = lilka::resources.loadImage("image.bmp");
if (!image) {
  Serial.println("Failed to load image");
  return;
}
lilka::Image *rotatedImage = new lilka::Image(image->width, image->height);
// Повертаємо на 30 градусів, заповнюючи пікселі, які виходять за межі
↪зображення, білим кольором:
image->rotate(30, rotatedImage, lilka::colors::White);
// Звільнюємо пам'ять
```

(continues on next page)

(continued from previous page)

```
delete image;
delete rotatedImage;
```

Дивись також:

Display::drawImageTransformed, Canvas::drawImageTransformed, Transform

> Попередження: dest повинен бути ініціалізований заздалегідь.

Параметри

- angle – Кут обертання в градусах.
- dest – Вказівник на Image, в яке буде записано обернуте зображення.
- blankColor – 16-бітний колір (5-6-5), який буде використаний для заповнення пікселів, які виходять за межі зображення.

void flipX(Image *dest)

Віддзеркалити зображення по горизонталі і записати результат в dest.

void flipY(Image *dest)

Віддзеркалити зображення по вертикалі і записати результат в dest.

Public Members

uint32_t width

Ширина зображення.

uint32_t height

Висота зображення.

int32_t transparentColor

16-бітний колір (5-6-5), який буде прозорим. За замовчуванням -1 (прозорість відсутня).

int16_t pivotX

Координата X центральної осі зображення.

int16_t pivotY

Координата Y центральної осі зображення.

uint16_t *pixels

Масив пікселів зображення. Ініціалізується в конструкторі автоматично.

Public Static Functions

static Image *newFromRLE(const uint8_t *data, uint32_t length, uint32_t width, uint32_t height, int32_t transparentColor = -1, int16_t pivotX = 0, int16_t pivotY = 0)

Створити зображення з масиву 16-бітних точок, стисненого алгоритмом RLE.

class Transform

Клас для роботи з афінними перетвореннями.

Афінні перетворення - це перетворення, які зберігають паралельність ліній. Вони включають в себе обертання, масштабування та віддзеркалення.

Перетворення - це всього лиш матриця 2x2. Застосування перетворення до вектора - це множення цього вектора на матрицю перетворення. Магія!

```
// Цей код обертає зображення на 30 градусів і тоді віддзеркалює його по
↪горизонталі
lilka::Transform transform = lilka::Transform().rotate(30).flipX();
lilka::display.drawImageTransformed(image, 32, 64, transform);
```

Public Functions

Transform()

Створити об'єкт класу lilka::Transform.

Transform rotate(int16_t angle)

Обернути навколо центру на кут angle (в градусах).

Оскільки на екрані вісь Y вказує вниз, обертання буде здійснено за годинниковою стрілкою.

Параметри
angle – Кут обертання в градусах.

Повертає
Нове перетворення.

Transform scale(float scaleX, float scaleY)

Масштабувати по X та Y.

Щоб віддзеркалити зображення, використайте від'ємні значення (наприклад, -1).

Параметри

- scaleX – Масштаб по X.
- scaleY – Масштаб по Y.

Повертає
Нове перетворення.

Transform multiply(Transform other)

Помножити це перетворення на інше.

Примітка: Зверніть увагу: при комбінації перетворень порядок важливий, і він є оберненим до порядку множення матриць! В результаті цього, перетворення other буде виконано перед поточним перетворенням. Тобто якщо в вас є деякі перетворення A та B, то перетворення A.multiply(B) утворить нове перетворення, в якому спочатку виконається перетворення B, а потім A.

Примітка: Для уникнення плутанини рекомендуємо використовувати більш високорівневі методи, такі як rotate та scale.

Параметри
other – Інше перетворення.

Повертає
Перетворення, яке є результатом застосування цього перетворення після other перетворення.

```
lilka::Transform rotate30 = lilka::Transform().rotate(30); // обернути
↪на 30 градусів
lilka::Transform scale2x = lilka::Transform().scale(2, 2); // збільшити
↪в 2 рази
lilka::Transform scaleThenRotate = rotate30.multiply(scale2x); //
↪результат - це перетворення "спочатку збільшити, а потім
↪обернути", а не навпаки!
```

Transform inverse()

Інвертувати перетворення.

Примітка: Інвертне перетворення - це таке перетворення, яке скасує це перетворення, тобто є зворотнім до цього.

Повертає
Інвертоване перетворення.

inline int_vector_t transform(int_vector_t vector)

Перетворити вектор, використовуючи це перетворення.

Параметри
vector – Вхідний вектор.

Повертає
Результат перетворення.

6.7 FileUtils: Допоміжні функції для роботи з файлами

FileUtils lilka::fileutils

Екземпляр класу FileUtils, який можна використовувати для роботи з дисплеєм. Вам не потрібно інстанціювати FileUtils вручну.

class FileUtils

Клас, що містить допоміжні функції для роботи з файловими системами.

Цей клас НЕ містить методів типу open, read, write і т.д., оскільки ці методи вже є в класах SD та SPIFFS і для роботи з файлами вам слід використовуйте саме їх, оскільки вони добре документовані та легкі в використанні: https://www.arduino.cc/reference/en/libraries/sd/

Тому у цьому класі ви знайдете лише допоміжні методи, такі як ініціалізація носіїв даних, перетворення шляхів, читання директорії, перевірка доступності файлових систем тощо.

Крім цього, деякий сторонній код (наприклад, емулятор NES чи рушій Doom) працюють з файлами через стандартні функції Ci, тому цей клас містить методи для перетворення канонічних шляхів в локальні та навпаки.

- Локальний шлях - це такий, який визначається відносно кореня носія даних (наприклад, /test.txt). В більшості випадків ви будуте працювати лише з локальними шляхами. Наприклад, File test = SD.open("/test.txt") відкриє файл test.txt з SD-карти.
- Канонічний шлях - це такий, який визначається відносно кореня віртуальної файлової системи ESP32 (наприклад, /sd/test.txt). Ви можете використовувати канонічні шляхи для взаємодії з файлами через стандартні функції Ci, наприклад FILE* test = fopen("/sd/test.txt", "r").

Детальніше про віртуальну файлову систему ESP32 можна прочитати тут: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/storage/vfs.html

Public Functions

void begin(bool beginSD = true, bool beginSPIFFS = true)

Ініціалізувати SPIFFS та SD-карту

> Попередження: Цей метод викликається автоматично при виклику lilka::begin().

bool initSD()

Ініціалізувати SD-карту

void initSPIFFS()

Ініціалізує SPIFFS.

uint32_t getEntryCount(FS *driver, const String &localPath)

Кількість елементів у директорії

Параметри

- driver – драйвер файлової системи
- localPath – шлях до директорії

Повертає
кількість елементів

const String getSDRoot()

Канонічний шлях до SD

Повертає
канонічний шлях (наприклад, /sd)

const String getSPIFFSRoot()

Канонічний шлях до SPIFFS

Повертає
канонічний шлях (наприклад, /spiffs)

bool isSDAvailable()

Перевірити, чи доступна SD картка

Повертає
true у разі успіху

bool isSPIFFSAvailable()

Перевірити, чи доступна SPIFFS

Повертає
true у разі успіху

const String getCannonicalPath(const FS *driver, const String &localPath)

Перетворити локальний шлях в канонічний

Приклад використання:

```
String localPath = "/test.txt";
String cannonicalPath = fileutils.getCannonicalPath(&SD, localPath);
// cannonicalPath міститиме рядок "/sd/test.txt"
FILE* test = fopen(cannonicalPath.c_str(), "r"); // Відкрити файл "/sd/test.txt
↪" з SD-карти
```

Параметри

- driver – драйвер файлової системи
- localPath – локальний шлях

Повертає
канонічний шлях

const PathInfo getLocalPathInfo(const String &cannonicalPath)

Отримати інформацію про канонічний шлях

Використовується для перетворення канонічного шляху в локальний або для визначення драйвера файлової системи.

Приклад використання:

```
String cannonicalPath = "/sd/test.txt";
PathInfo pathInfo = fileutils.getLocalPathInfo(cannonicalPath);
// pathInfo.driver міститиме вказівник на драйвер файлової системи SD
// pathInfo.path міститиме рядок "/test.txt"
File test = pathInfo.driver->open(pathInfo.path); // Відкрити файл "/test.txt" з
↪SD-карти
```

Параметри
cannonicalPath – локальний шлях

Повертає
структуру PathInfo

size_t listDir(FS *driver, const String &localPath, Entry entries[])

Отримати елементи в директорії за відносним шляхом

Параметри

- driver – драйвер файлової системи (наприклад, &SD)
- localPath – локальний шлях
- entries – вказівник, за яким буде записано елементи

Повертає
кількість записаних елементів

const String joinPath(const String &lPath, const String &rPath)

Об'єднує шляхи

Параметри

- lPath – ліва половина шляху
- rPath – права половина шляху

Повертає
повний шлях

bool createSDPartTable()

Створити нову таблицю розділів на SD картці

> Попередження: Після виклику цієї функції необхідно перезавантажити систему

Повертає
true у разі успіху

bool formatSD()

Форматувати SD картку

> Попередження: Після виклику цієї функції необхідно перезавантажити систему

Повертає
true у разі успіху

const String stripPath(const String &path)

Обрізати слеші в кінці шляху.

Параметри
path – Шлях до файлу

Повертає
Шлях без кінцевих слешів

const String getHumanFriendlySize(const size_t size, bool compact = false)

Повернути розмір в читабельному форматі (наприклад, 101 MB).

```
fileutils.getHumanFriendlySize(1234567); // Поверне "1.23 MB"
```

Параметри

- size – Розмір (в байтах)
- compact – Чи виводити розмір компактно (наприклад, 1.23MB замість 1.23 MB)

Повертає
Рядок, що містить читабельний розмір з суфіксами одиниць виміру

struct PathInfo

Інформація про шлях

Public Members

FS *driver

Вказівник на драйвер файлової системи

String path

Локальний шлях у цій файловій системі

struct Entry

Інформація про елемент в директорії

Public Members

String name

Ім'я елементу

EntryType type

Тип елементу

size_t size

Розмір елементу

enum lilka::EntryType

Тип елементу в директорії

Values:

enumerator ENT_FILE

Файл

enumerator ENT_DIRECTORY

Директорія

6.8 MultiBoot: Завантажувач прошивок

MultiBoot lilka::multiboot

Екземпляр класу MultiBoot, який можна використовувати для роботи з завантажувачем. Вам не потрібно інстанціювати MultiBoot вручну.

class MultiBoot

Завантажувач прошивок з microSD-картки.

Дозволяє прочитати файл прошивки з microSD-картки в OTA-розділ та запустити його один раз, не замінюючи поточну прошивку.

Нова прошивка зберігається в OTA-розділі та запускається при перезавантаженні, але активною залишається основна прошивка (app rollback).

Це дозволяє компілювати за записувати різні прошивки на microSD-карту і запускати їх через браузер SD-карти без необхідності перепрошивки.

Детальніше про auto-rollback: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/ota.html#rollback-process

Приклад використання:

```
#include <lilka.h>

void setup() {
    lilka::begin();

    Serial.println("Завантаження прошивки 'firmware.bin' з microSD-картки...");

    lilka::multiboot.start("/sd/firmware.bin"); // Почати завантаження
    while (lilka::multiboot.process() != 0) { // Обробити завантаження
        Serial.println("Завантажено " + String(lilka::multiboot.getBytesWritten()) + "/
↪" + String(lilka::multiboot.getBytesTotal()) + " байтів");
    }
    lilka::multiboot.finishAndReboot(); // Завершити завантаження та␣
↪перезавантажити пристрій
}
```

Public Functions

MultiBoot()

void begin()

Ініціалізувати завантажувач.

> Попередження: Цей метод викликається автоматично при виклику lilka::begin().

int start(String path)

Почати завантаження.

Повертає
0, якщо завантаження почалося успішно, <0 - у разі помилки.

int process()

Обробити завантаження. Цей метод повинен викликатися в циклі, поки не поверне 0. Щоразу він опрацьовує частину файлу та записує її в OTA-розділ.

Повертає
0, якщо завантаження завершилося успішно, <0 - у разі помилки, >0 - означає кількість байтів, які було оброблено.

void cancel()

Перервати завантаження.

int getBytesWritten()

Отримати кількість байтів, які було записано в OTA-розділ.

Повертає
Кількість байтів

int getBytesTotal()

Отримати загальну кількість байтів, які потрібно записати в OTA-розділ.

Повертає
Кількість байтів

int finishAndReboot()

Завершити завантаження та перезавантажити пристрій.

Повертає
<0 - у разі помилки. В разі успіху цей метод не повертається, оскільки пристрій перезавантажується.

String getFirmwarePath()

6.9 Resources: Ресурси

Resources lilka::resources

Екземпляр класу Resources, який можна використовувати для завантаження ресурсів. Вам не потрібно інстанціювати Resources вручну.

class Resources

Клас для роботи з ресурсами - зображенням, файлами даних тощо.

Public Functions

Image *loadImage(String filename, int32_t transparentColor = -1, int32_t pivotX = 0, int32_t pivotY = 0)

Завантажити зображення в форматі BMP з файлу.

Приклад:

```
lilka::Image *image = lilka::resources.loadImage("image.bmp", lilka::colors::Yellow);
↪ //
Жовтий колір буде прозорим if (!image) {
    Serial.println("Failed to load image");
    return;
}
// Відобразити зображення на екрані
lilka::display.drawImage(image, 50, 100)
// Або:
lilka::display.draw16bitRGBBitmapWithTranColor(50, 100, image->pixels, image->
↪transparentColor, image->width, image->height);
// Звільнити пам'ять
delete image;
```

> Попередження: Пам'ять для зображення виділяється динамічно. Після використання зображення, його потрібно видалити за допомогою delete.

Параметри

- filename – Шлях до файлу.
- transparentColor – 16-бітний колір (5-6-5), який буде прозорим. За замовчуванням -1 (прозорість відсутня).
- pivotX – X-координата точки, яка буде центром зображення. За замовчуванням 0.
- pivotY – Y-координата точки, яка буде центром зображення. За замовчуванням 0.

Повертає
Вказівник на зображення.

int readFile(String filename, String &fileContent)

Прочитати вміст файлу.

TODO: Update sdcard/filesystem stuff

> Попередження: Не використовуйте цей метод для читання великих файлів, оскільки весь вміст файлу зберігається в пам'яті. Для великих файлів використовуйте методи sdcard та filesystem.

Параметри

- filename – Шлях до файлу.
- fileContent – Змінна, в яку буде записано вміст файлу.

Повертає
0, якщо читання успішне; -1, якщо файл не знайдено

int writeFile(String filename, String fileContent)

Записати вміст файлу.

Параметри

- filename – Шлях до файлу.
- fileContent – Вміст файлу.

Повертає
0, якщо запис успішний; -1, якщо запис не вдався

6.10 SPI: Шина SPI

SPIClass lilka::SPI1(SPI1_NUM)

Системна шина SPI. Використовується внутрішньо для роботи з дисплеєм та SD-картою. Ми радимо не чіпати цей об'єкт, якщо ви не знаєте, що робите.

SPIClass lilka::SPI2(SPI2_NUM)

Користувацька шина SPI.

Використовується для роботи з будь-якими іншими пристроями. Для її використання потрібно викликати SPI2.begin(). Приклад використання SPI2 для роботи з двома SPI-пристроями:

```
#include "lilka.h"

// Визначення пінів для SPI2. Можна використовувати будь-які піни,
// які виведені на роз'єм розширення:
#define SPI2_SCK 12
#define SPI2_MISO 13
#define SPI2_MOSI 14
#define SPI2_DEV1_CS 21 // Chip Select для пристрою 1
#define SPI2_DEV2_CS 47 // Chip Select для пристрою 2

void setup() {
    lilka::begin();
    lilka::SPI2.begin(SPI2_SCK, SPI2_MISO, SPI2_MOSI);
    pinMode(SPI2_DEV1_CS, OUTPUT);
    pinMode(SPI2_DEV2_CS, OUTPUT);
    digitalWrite(SPI2_DEV1_CS, HIGH); // Вимкнути пристрій 1
    digitalWrite(SPI2_DEV2_CS, HIGH); // Вимкнути пристрій 2
}

void loop() {
    // Надсилання даних пристроєві, під'єднаному до SPI2 (лінія активації - SPI2_
↪DEV1_CS)
    lilka::SPI2.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));
    digitalWrite(SPI2_DEV1_CS, LOW); // Увімкнути пристрій 1
    lilka::SPI2.transfer(0x55);
    digitalWrite(SPI2_DEV1_CS, HIGH); // Вимкнути пристрій 1
    lilka::SPI2.endTransaction();

    delay(500);

    // Надсилання даних пристроєві, під'єднаному до SPI2 (лінія активації - SPI2_
↪DEV2_CS)
    lilka::SPI2.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));
    digitalWrite(SPI2_DEV2_CS, LOW); // Увімкнути пристрій 2
    lilka::SPI2.transfer(0xAA);
    digitalWrite(SPI2_DEV2_CS, HIGH); // Вимкнути пристрій 2
    lilka::SPI2.endTransaction();
```

(continues on next page)

(continued from previous page)

```
    delay(1000);
}
```

6.11 UI: Інтерфейс користувача

Цей модуль містить класи, які відповідають за відображення простих елементів інтерфейсу користувача: меню, сповіщення тощо.

class Menu

Клас для відображення меню.

Дозволяє відобразити меню з пунктами, які можна вибирати за допомогою стрілок вгору/вниз та підтвердити вибір кнопкою A.

Приклад використання:

```
#include <lilka.h>

void setup() {
    lilka::begin();
}

void loop() {
    lilka::Menu dreams("Оберіть щось");
    dreams.addItem("Смерть русні");
    dreams.addItem("Ядерка на червону площу");
    dreams.addItem("Повернення Криму");
    while (!dreams.isFinished()) {
        dreams.update();
        dreams.draw(&lilka::display);
    }
    int index = dreams.getCursor();
    lilka::MenuItem item;
    dreams.getItem(index, &item);
    Serial.println(String("Ви обрали пункт ") + item.title);
}
```

Public Functions

explicit Menu(String title = "Меню")

Конструктор класу.

Параметри
title – Заголовок меню.

void setTitle(String title)

Встановити новий заголовок меню

Параметри
title – Заголовок меню.

void addItem(String title, const menu_icon_t *icon = 0, uint16_t color = 0, String postfix = "")

Додати пункт до меню.

Параметри

- title – Заголовок пункту.
- icon – Іконка пункту (масив з uint16_t розміром 576 елементів, який представляє 24x24px зображення). За замовчуванням 0 (відсутня іконка).
- color – Колір пункту. За замовчуванням 0 (стандартний колір).
- postfix – Текст, який додається після заголовка пункту і вирівнюється до правого краю меню.

void setCursor(int16_t cursor)

Встановити курсор на пункт меню.

Параметри
cursor – Індекс пункту меню.

void update()

Оновити стан меню.

Цю функцію потрібно викликати, щоб меню опрацювало вхідні дані від користувача та оновило свій стан.

void draw(Arduino_GFX *canvas)

Намалювати меню на Display або Canvas.

Приклад використання:

```
// ...
menu.draw(&lilka::display); // намалювати меню на Display
menu.draw(&canvas); // намалювати меню на Canvas
// ...
```

Параметри
canvas – Вказівник на Display або Canvas, на якому потрібно намалювати меню.

bool isFinished()

Перевірити, чи обрано пункт меню.

Якщо пункт обрано, тобто користувач натиснув кнопку «A» (або іншу кнопку, яка була додана за допомогою addActivationButton()), повертається true, інакше false.

bool setItem(int16_t index, String title, const menu_icon_t *icon = 0, uint16_t color = 0, String postfix = "")

Змінити пункт меню

Повертає значення true, якщо пункт було змінено

Параметри

- index – Індекс пункту.
- title – Заголовок пункту.
- icon – Іконка пункту (масив з uint16_t розміром 576 елементів, який представляє 24x24px зображення). За замовчуванням 0 (відсутня іконка).
- color – Колір пункту. За замовчуванням 0 (стандартний колір).
- postfix – Текст, який додається після заголовка пункту і вирівнюється до правого краю меню.

bool getItem(int16_t index, MenuItem *menuItem)

Отримати пункт меню

Параметри

- index – Індекс пункту.
- menuItem – Вказівник на lilka::MenuItem куди буде скопійовано пункт Повертає true у разі успіху

int16_t getCursor()

Отримати індекс обраного пункту меню.

void clearItems()

Очистити меню і зробити його доступним для повторного використання

int16_t getItemCount()

Отримати кількість пунктів меню.

Повертає
Кількість пунктів меню.

void addActivationButton(Button activationButton)

Дозволити вибір пункту меню за допомогою інших кнопок.

За замовчуванням вибір пункту можливий тільки за допомогою кнопки «A». Після виклику цієї функції можна вибирати пункт за допомогою додаткових кнопок.

Дивись також:

getButton

Button getButton()

Отримати кнопку, якою користувач обрав пункт меню.

Якщо пункт не обрано, результат буде невизначеним. Рекомендується використовувати цю функцію тільки після того, як isFinished() поверне true.

class Alert

Клас для відображення сповіщення.

Дозволяє відобразити сповіщення, яке користувач може закрити кнопкою A.

Приклад використання:

```
#include <lilka.h>

void setup() {
    lilka::begin();
}

void loop() {
    lilka::Alert warning("Увага", "Повітряна тривога в москві, загроза балістичних
↪ракет!");
    warning.draw(&lilka::display);
    while (!warning.isFinished()) {
        warning.update();
    }
}
```

Public Functions

Alert(String title, String message)

Конструктор класу.

Параметри

- title – Заголовок сповіщення.
- message – Повідомлення сповіщення.

void setTitle(String title)

Змінити заголовок сповіщення.

Параметри

title – Новий заголовок сповіщення.

void setMessage(String message)

Змінити повідомлення сповіщення.

Параметри

message – Нове повідомлення сповіщення.

void update()

Оновити стан сповіщення.

Цю функцію потрібно викликати, щоб сповіщення опрацювало вхідні дані від користувача та оновило свій стан.

void draw(Arduino_GFX *canvas)

Намалювати сповіщення на Display або Canvas.

Приклад використання:

```
// ...
alert.draw(&lilka::display); // намалювати сповіщення на Display
alert.draw(&canvas); // намалювати сповіщення на Canvas
// ...
```

Параметри
canvas – Вказівник на Display або Canvas, на якому потрібно намалювати сповіщення.

bool isFinished()

Перевірити, чи користувач закрив сповіщення.

Якщо сповіщення закрито, тобто користувач натиснув кнопку «A» (або іншу кнопку, яка була додана за допомогою addActivationButton()), повертається true, інакше false.

void addActivationButton(Button activationButton)

Дозволити закриття сповіщення за допомогою інших кнопок.

За замовчуванням закриття сповіщення можливе тільки за допомогою кнопки «A». Після виклику цієї функції можна закривати сповіщення за допомогою додаткових кнопок.

Дивись також:

getButton

Button getButton()

Отримати кнопку, якою користувач закрив сповіщення.

Якщо сповіщення не закрито, результат буде невизначеним. Рекомендується використовувати цю функцію тільки після того, як isFinished() поверне true.

class ProgressDialog

Клас для відображення індикатора виконання.

Дозволяє відобразити індикатор виконання, який можна оновлювати та змінювати його повідомлення.

Приклад використання:

```
#include <lilka.h>

void setup() {
    lilka::begin();
}

void loop() {
    lilka::ProgressDialog progress("Почекайте", "Завантаження даних...");
    for (int i = 0; i <= 100; i++) {
        progress.setProgress(i);
        progress.draw(&lilka::display);
```

(continues on next page)

(continued from previous page)

```
        delay(100);
    }
    progress.setMessage("Дані завантажено!");
    delay(1000);
}
```

Public Functions

ProgressDialog(String title, String message)

Конструктор класу.

Параметри

- title – Заголовок індикатора виконання.
- message – Повідомлення індикатора виконання.

void setProgress(int16_t progress)

Встановити прогрес виконання.

Параметри

progress – Прогрес виконання від 0 до 100.

void setMessage(String message)

Встановити повідомлення.

Параметри

message – Повідомлення.

void draw(Arduino_GFX *canvas)

Намалювати індикатор виконання на Display або Canvas .

Приклад використання:

```
// ...
progress.draw(&lilka::display); // намалювати індикатор виконання на Display
progress.draw(&canvas); // намалювати індикатор виконання на Canvas
// ...
```

Параметри

canvas – Вказівник на Display або Canvas, на якому потрібно намалювати індикатор виконання.

class InputDialog

Клас для відображення діалогового вікна введення.

Малює вікно введення та екранну клавіатуру, дозволяє вводити текст та підтверджувати введення.

Приклад використання:

```
InputDialog dialog("Введіть пароль");
dialog.setMasked(true);
dialog.setValue("1234");
while (!dialog.isFinished()) {
    dialog.update();
    dialog.draw(&lilka::display);
}
String password = dialog.getValue();
```

Public Functions

explicit InputDialog(String title)

Конструктор класу.

Параметри
title – Заголовок діалогового вікна введення.

void setMasked(bool masked)

Встановити маскування введеного тексту. Якщо встановлено true, введений текст буде відображатися як зірочки.

Параметри
masked – Чи маскувати введений текст.

void setValue(String value)

Встановити початкове значення введеного тексту.

Параметри
value – Текст.

void update()

Оновити стан діалогового вікна введення.

Цю функцію потрібно викликати, щоб діалогове вікно введення опрацювало вхідні дані від користувача та оновило свій стан.

void draw(Arduino_GFX *canvas)

Намалювати діалогове вікно введення на Display або Canvas.

Параметри
canvas – Вказівник на Display або Canvas, на якому потрібно намалювати діалогове вікно введення.

bool isFinished()

Перевірити, чи користувач завершив введення тексту.

Якщо введення завершено, тобто користувач натиснув кнопку «START», повертається true, інакше false.

6.12 fmath: Швидкі математичні функції

Насправді, в бібліотеках Arduino та ESP32 вже є чимало математичних функцій, і деякі з них навіть мають апаратне прискорення. Проте Лілка додає ще декілька швидких та корисних функцій.

float lilka::fSin360(int deg)

Швидке обчислення синуса.

Ця функція працює значно швидше, ніж стандартна функція синуса, але лише для цілих значень кута.

Параметри
deg – Кут в градусах.

Повертає
Значення синуса кута.

float lilka::fCos360(int deg)

Швидке обчислення косинуса.

Ця функція працює значно швидше, ніж стандартна функція косинуса, але лише для цілих значень кута.

Параметри
deg – Кут в градусах.

Повертає
Значення косинуса кута.

float lilka::fSin32(int fract)

Швидке обчислення синуса для кутів, кратних 11.25 градусів (тобто для кола, поділеного на 32 сектори).

```
float a = fSin32(1); // Те саме, що і ``sin(11.25°)``
float b = fSin32(8); // Те саме, що і ``sin(90°)``
float c = fSin32(16); // Те саме, що і ``sin(180°)``
```

Параметри
fract – Кут в градусах, поділений на 32.

Повертає
Значення синуса кута.

float lilka::fCos32(int fract)

Швидке обчислення косинуса для кутів, кратних 11.25 градусів (тобто для кола, поділеного на 32 сектори).

Параметри
fract – Кут в градусах, поділений на 32.

Повертає
Значення косинуса кута.

6.13 colors: 16-бітні кольори

enum lilka::colors::RGB565_Colors

Values:

enumerator Absolute_zero

0x0257

enumerator Acid_green

0xADE3

enumerator Aero

0x7DDC

enumerator Aero_blue

0xBF3A

enumerator African_violet

0xB437

enumerator Air_force_blue

0x5C54

enumerator Air_superiority_blue

0x7517

enumerator Alabaster

0xEF5B

enumerator Alice_blue

0xEFBF

enumerator Alloy_orange

0xC302

enumerator Almond

0xEEF9

enumerator Amaranth

0xE16A

enumerator Amaranth_mp

0x996D

enumerator Amaranth_pink
0xECF7

enumerator Amaranth_purple
0xA94A

enumerator Amaranth_red
0xD105

enumerator Amazon
0x3BCB

enumerator Amber
0xFDE0

enumerator Amber_saeece
0xFBE0

enumerator Amethyst
0x9B39

enumerator Android_green
0xA627

enumerator Antique_brass
0xCCAE

enumerator Antique_bronze
0x62E4

enumerator Antique_fuchsia
0x92F0

enumerator Antique_ruby
0x80E5

enumerator Antique_white
0xF75A

enumerator Ao_english
0x0400

enumerator Apple_green
0x8DA0

enumerator Apricot
0xFE76

enumerator Aqua
0x07FF

enumerator Aquamarine
0x7FFA

enumerator Arctic_lime
0xCFE2

enumerator Argentinian_blue
0x6D9D

enumerator Army_green
0x4AA4

enumerator Artichoke
0x8CAF

enumerator Artichoke_green_pantone
0x4B68

enumerator Arylide_yellow
0xE6AD

enumerator Ash_gray
0xB5F6

enumerator Asparagus
0x854D

enumerator Atomic_tangerine
0xFCCC

enumerator Auburn
0xA145

enumerator Aureolin
0xFF60

enumerator Avocado
0x5400

enumerator Azure
0x041F

enumerator Azure_web
0xEFFF

enumerator Azure_x11web_color
0xEFFF

enumerator Baby_blue
0x8E7D

enumerator Baby_blue_eyes
0xA65D

enumerator Baby_pink
0xF618

enumerator Baby_powder
0xFFFE

enumerator Baker_miller_pink
0xFC95

enumerator Banana_mania
0xF736

enumerator Barbie_pink
0xD8D0

enumerator Barn_red
0x7840

enumerator Battleship_gray
0x8430

enumerator Battleship_grey
0x8430

enumerator Bdazzled_blue
0x32D2

enumerator Bean
0x3860

enumerator Beau_blue
0xBE9C

enumerator Beaver
0x9C0E

enumerator Beige
0xF7BB

enumerator Berkeley_blue
0x018C

enumerator Big_dip_oruby
0x9928

enumerator Bisque
0xFF18

enumerator Bistre
0x3964

enumerator Bistre_brown
0x9383

enumerator Bitter_lemon
0xCEE2

enumerator Bitter_lime
0xBFE0

enumerator Bittersweet
0xFB6B

enumerator Bittersweet_shimmer
0xBA8A

enumerator Black
0x0000

enumerator Black_bean
0x3860

enumerator Black_chocolate
0x18C2

enumerator Black_coffee
0x3986

enumerator Black_coral
0x530D

enumerator Black_olive
0x39E7

enumerator Black_shadows
0xBD76

enumerator Blanched_almond
0xFF59

enumerator Blast_off_bronze
0xA38C

enumerator Bleu_de_france
0x347C

enumerator Blizzard_blue
0xAF3D

enumerator Blond
0xF777

enumerator Blood_red
0x6000

enumerator Blue
0x001F

enumerator Blue_bell
0xA519

enumerator Blue_cmyk_pigment_blue
0x31B3

enumerator Blue_computer_web_color
0x001F

enumerator Blue_crayola
0x23BF

enumerator Blue_gray
    0x64D9

enumerator Blue_green
    0x14D7

enumerator Blue_green_color_wheel
    0x0A68

enumerator Blue_jeans
    0x5D7D

enumerator Blue_munsell
    0x0495

enumerator Blue_ncs
    0x0437

enumerator Blue_pantone
    0x00D4

enumerator Blue_pigment
    0x31B3

enumerator Blue_ryb
    0x025F

enumerator Blue_sapphire
    0x1310

enumerator Blue_violet
    0x897B

enumerator Blue_violet_color_wheel
    0x48CF

enumerator Blue_violet_crayola
    0x7337

enumerator Blue_yonder
    0x5394

enumerator Bluetiful
    0x3B5C

enumerator Blush
0xDAF0

enumerator Bole
0x7A27

enumerator Bone
0xE6D8

enumerator Bottle_green
0x0349

enumerator Brandeis_blue
0x039F

enumerator Brandy
0x8208

enumerator Brick_red
0xCA0A

enumerator Bright_green
0x67E0

enumerator Bright_lilac
0xD49D

enumerator Bright_maroon
0xC109

enumerator Bright_navy_blue
0x1BBA

enumerator Bright_yellow_crayola
0xFD44

enumerator Brilliant_rose
0xFAB4

enumerator Brink_pink
0xFB0F

enumerator British_racing_green
0x0204

enumerator Bronze
0xCBE6

enumerator Brown
0x9260

enumerator Brown_sugar
0xAB69

enumerator Brunswick_green
0x1A68

enumerator Bubblegum_pink
0xF412

enumerator Bud_green
0x7DAC

enumerator Buff
0xDD0D

enumerator Burgundy
0x8004

enumerator Burlywood
0xDDB0

enumerator Burnished_brown
0xA3CE

enumerator Burnt_orange
0xBAA0

enumerator Burnt_sienna
0xE3AA

enumerator Burnt_umber
0x89A4

enumerator Butterscotch
0xDCA8

enumerator Byzantine
0xB9B4

enumerator Byzantium
: 0x714C

enumerator Cadet
: 0x534E

enumerator Cadet_blue
: 0x64F3

enumerator Cadet_blue_crayola
: 0xAD98

enumerator Cadet_gray
: 0x9515

enumerator Cadet_grey
: 0x9515

enumerator Cadmium_green
: 0x0347

enumerator Cadmium_orange
: 0xEC25

enumerator Cadmium_red
: 0xE004

enumerator Cadmium_yellow
: 0xFFA0

enumerator Cafe_au_lait
: 0xA3CB

enumerator Cafe_noir
: 0x49A4

enumerator Cambridge_blue
: 0xA615

enumerator Camel
: 0xBCCD

enumerator Cameo_pink
: 0xEDD9

enumerator Canary
0xFFF3

enumerator Canary_yellow
0xFFF3

enumerator Candy_apple_red
0xF840

enumerator Candy_pink
0xE38F

enumerator Cantaloupe_melon
0xFDD6

enumerator Capri
0x05FF

enumerator Caput_mortuum
0x5944

enumerator Cardinal
0xC0E7

enumerator Caribbean_current
0x036D

enumerator Caribbean_green
0x0653

enumerator Carmine
0x9003

enumerator Carmine_mp
0xD008

enumerator Carnation_pink
0xFD38

enumerator Carnelian
0xB0E3

enumerator Carolina_blue
0x551A

enumerator Carrot_orange
0xEC84

enumerator Castleton_green
0x02A8

enumerator Catawba
0x71A8

enumerator Cedar_chest
0xC2C9

enumerator Celadon
0xAF15

enumerator Celadon_blue
0x03D4

enumerator Celadon_green
0x342F

enumerator Celeste
0xB7FF

enumerator Celestial_blue
0x4CB9

enumerator Celtic_blue
0x2359

enumerator Cerise
0xD98C

enumerator Cerulean
0x03D4

enumerator Cerulean_blue
0x2A97

enumerator Cerulean_crayola
0x255A

enumerator Cerulean_frost
0x6CD8

enumerator Cg_blue
0x03D4

enumerator Cg_red
0xD9E6

enumerator Champagne
0xF739

enumerator Champagne_pink
0xEEF9

enumerator Charcoal
0x3A2A

enumerator Charleston_green
0x2165

enumerator Charm_pink
0xE475

enumerator Chartreuse_traditional
0xDFE0

enumerator Chartreuse_web
0x7FE0

enumerator Chefchaouen_blue
0x4C7C

enumerator Cherry_blossom_pink
0xFDB8

enumerator Chestnut
0x9226

enumerator Chilean_pink
0xE617

enumerator Chili_red
0xD9E5

enumerator China_pink
0xDB74

enumerator China_rose
0xA28D

enumerator Chinese_red
0xA9C4

enumerator Chinese_violet
0x8311

enumerator Chinese_yellow
0xFD80

enumerator Chocolate
0x7A00

enumerator Chocolate_cosmos
0x5883

enumerator Chocolate_traditional
0x7A00

enumerator Chocolate_web
0xD344

enumerator Chrome_yellow
0xFD20

enumerator Cinereous
0x940F

enumerator Cinnabar
0xE266

enumerator Cinnamon_satin
0xCB0F

enumerator Citrine
0xE661

enumerator Citron
0x9D44

enumerator Claret
0x78C6

enumerator Cobalt_blue
0x0255

enumerator Cocoa_brown
0xD344

enumerator Coffee
0x6A67

enumerator Columbia_blue
0xB6DD

enumerator Congo_pink
0xF40F

enumerator Cool_gray
0x9497

enumerator Cool_grey
0x8C95

enumerator Copper
0xB386

enumerator Copper_crayola
0xDC4D

enumerator Copper_penny
0xAB6D

enumerator Copper_red
0xCB6A

enumerator Copper_rose
0x9B2C

enumerator Coquelicot
0xF9C0

enumerator Coral
0xFBEA

enumerator Coral_pink
0xF40F

enumerator Cordovan
0x8A08

enumerator Corn
0xFF4B

enumerator Cornell_red
0xB0E3

enumerator Cornflower_blue
0x64BD

enumerator Cornsilk
0xFFBB

enumerator Cosmic_cobalt
0x3171

enumerator Cosmic_latte
0xFFBC

enumerator Cotton_candy
0xFDDA

enumerator Coyote_brown
0x8307

enumerator Cream
0xFFF9

enumerator Crimson
0xD8A7

enumerator Crimson_ua
0x98E6

enumerator Crystal
0xA6BB

enumerator Cultured
0xF7BE

enumerator Cyan
0x07FF

enumerator Cyan_additive_secondary
0x07FF

enumerator Cyan_process
0x05BD

enumerator Cyan_subtractive_primary
0x05BD

enumerator Cyber_grape
0x5A0F

enumerator Cyber_yellow
0xFE80

enumerator Cyclamen
0xF374

enumerator Dark_blue
0x0011

enumerator Dark_blue_gray
0x6333

enumerator Dark_brown
0x5A06

enumerator Dark_byzantium
0x59CA

enumerator Dark_cornflower_blue
0x2A11

enumerator Dark_cyan
0x0451

enumerator Dark_electric_blue
0x534F

enumerator Dark_goldenrod
0xB421

enumerator Dark_green
0x0184

enumerator Dark_green_x11
0x0320

enumerator Dark_jungle_green
0x1924

enumerator Dark_khaki
0xBDAD

enumerator Dark_lava
0x49E6

enumerator Dark_liver
0x526A

enumerator Dark_liver_horses
0x51E7

enumerator Dark_magenta
0x8811

enumerator Dark_moss_green
0x4AE4

enumerator Dark_olive_green
0x5346

enumerator Dark_orange
0xFC60

enumerator Dark_orchid
0x9999

enumerator Dark_pastel_green
0x05E7

enumerator Dark_purple
0x30C6

enumerator Dark_red
0x8800

enumerator Dark_salmon
0xE4AF

enumerator Dark_sea_green
0x8DD1

enumerator Dark_sienna
0x38A2

enumerator Dark_sky_blue
0x8DFA

enumerator Dark_slate_blue
0x49F1

enumerator Dark_slate_gray
0x328A

enumerator Dark_spring_green
0x1B88

enumerator Dark_turquoise
0x0679

enumerator Dark_violet
0x901A

enumerator Dartmouth_green
0x0387

enumerator Davys_gray
0x52AA

enumerator Davys_grey
0x52AA

enumerator Deep_cerise
0xD990

enumerator Deep_champagne
0xF6B4

enumerator Deep_chestnut
0xB269

enumerator Deep_jungle_green
0x0269

enumerator Deep_pink
0xF8B2

enumerator Deep_saffron
0xFCC6

enumerator Deep_sky_blue
0x05FF

enumerator Deep_space_sparkle
0x4B2D

enumerator Deep_taupe
0x7AEC

enumerator Delft_blue
0x218B

enumerator Denim
0x1B17

enumerator Denim_blue
0x2236

enumerator Desert
0xBCCD

enumerator Desert_sand
0xEE55

enumerator Dim_gray
0x6B4D

enumerator Dodger_blue
0x249F

enumerator Dogwood_rose
0xD0CD

enumerator Drab
0x9383

enumerator Duck_blue
0x03B2

enumerator Duke_blue
    0x0013

enumerator Dutch_white
    0xEEF7

enumerator Earth_yellow
    0xDD4C

enumerator Ebony
    0x52EA

enumerator Ecru
    0xC590

enumerator Eerie_black
    0x18E3

enumerator Eggplant
    0x620A

enumerator Eggshell
    0xEF5A

enumerator Egyptian_blue
    0x11B4

enumerator Eigengrau
    0x18A4

enumerator Electric_blue
    0x7FDF

enumerator Electric_green
    0x07E0

enumerator Electric_indigo
    0x681F

enumerator Electric_lime
    0xCFE0

enumerator Electric_purple
    0xB81F

enumerator Electric_violet
0x881F

enumerator Emerald
0x562F

enumerator Eminence
0x6990

enumerator English_green
0x1A68

enumerator English_lavender
0xB412

enumerator English_red
0xAA6A

enumerator English_vermillion
0xCA49

enumerator English_violet
0x51EB

enumerator Erin
0x07E8

enumerator Eton_blue
0x9634

enumerator Fairy_tale
0xEE19

enumerator Fallow
0xBCCD

enumerator Falu_red
0x80C3

enumerator Fandango
0xB1B1

enumerator Fandango_pink
0xDA90

enumerator Fashion_fuchsia
0xF014

enumerator Fawn
0xE54E

enumerator Feldgrau
0x4AEA

enumerator Fern
0x75CF

enumerator Fern_green
0x53C8

enumerator Field_drab
0x6AA4

enumerator Fiery_rose
0xFAAE

enumerator Fire_brick
0xB104

enumerator Fire_engine_red
0xC905

enumerator Fire_opal
0xE2E9

enumerator Firebrick
0xB104

enumerator Flame
0xDAC4

enumerator Flax
0xEED0

enumerator Flirt
0xA00D

enumerator Floral_white
0xFFDD

enumerator Fluorescent_blue
0x1F9D

enumerator Forest_green
0x2444

enumerator Forest_green_crayola
0x652E

enumerator Forest_green_traditional
0x0224

enumerator Forest_green_web
0x2444

enumerator French_beige
0xA3CB

enumerator French_bistre
0x8369

enumerator French_blue
0x0397

enumerator French_fuchsia
0xFA12

enumerator French_lilac
0x8311

enumerator French_lime
0x9FE7

enumerator French_mauve
0xD39A

enumerator French_pink
0xFB73

enumerator French_raspberry
0xC169

enumerator French_rose
0xF251

enumerator French_sky_blue
    0x75BF

enumerator French_violet
    0x8839

enumerator Frostbite
    0xE1B4

enumerator Fuchsia
    0xF81F

enumerator Fuchsia_crayola
    0xBAB7

enumerator Fuchsia_purple
    0xC9CF

enumerator Fuchsia_rose
    0xC22E

enumerator Fulvous
    0xE420

enumerator Fuzzy_wuzzy
    0x8204

enumerator Gainsboro
    0xDEDB

enumerator Gamboge
    0xE4C2

enumerator Garnet
    0x71A6

enumerator Generic_viridian
    0x03EC

enumerator Ghost_white
    0xF7BF

enumerator Giants_orange
    0xFAC4

enumerator Glaucous
0x6416

enumerator Glossy_grape
0xAC96

enumerator Go_green
0x054C

enumerator Gold
0xA3E0

enumerator Gold_crayola
0xE5F1

enumerator Gold_fusion
0x83A9

enumerator Gold_golden
0xFEA0

enumerator Gold_metallic
0xD567

enumerator Gold_web_golden
0xFEA0

enumerator Golden_brown
0x9B23

enumerator Golden_poppy
0xFE00

enumerator Golden_yellow
0xFEE0

enumerator Goldenrod
0xDD24

enumerator Granite_gray
0x6B2D

enumerator Granny_smith_apple
0xA713

enumerator Grape
0x6974

enumerator Gray_green
0x5B8D

enumerator Gray_web
0x8410

enumerator Gray_x11_gray
0xBDF7

enumerator Graygrey
0x8410

enumerator Green
0x07E0

enumerator Green_blue
0x1336

enumerator Green_blue_crayola
0x2C38

enumerator Green_crayola
0x1D4F

enumerator Green_cyan
0x04CC

enumerator Green_earth
0xDEF2

enumerator Green_htmlcss_color
0x0400

enumerator Green_lizard
0xA786

enumerator Green_munsell
0x054E

enumerator Green_ncs
0x04ED

enumerator Green_pantone
: 0x0568

enumerator Green_pigment
: 0x052A

enumerator Green_ryb
: 0x6566

enumerator Green_sheen
: 0x6D74

enumerator Green_web
: 0x0400

enumerator Green_x11_color_wheel
: 0x07E0

enumerator Green_yellow
: 0xAFE6

enumerator Green_yellow_crayola
: 0xEF32

enumerator Greenish_yellow
: 0xEF4C

enumerator Grullo
: 0xACD0

enumerator Gunmetal
: 0x29A7

enumerator Han_blue
: 0x4379

enumerator Han_purple
: 0x50DE

enumerator Hansa_yellow
: 0xE6AD

enumerator Harlequin
: 0x47E0

enumerator Harvest_gold
    0xDC80

enumerator Heat_wave
    0xFBC0

enumerator Heliotrope
    0xDB9F

enumerator Heliotrope_gray
    0xACD5

enumerator Hollywood_cerise
    0xF014

enumerator Honeydew
    0xEFFD

enumerator Honolulu_blue
    0x03B6

enumerator Hookers_green
    0x4BCD

enumerator Hot_magenta
    0xF8F9

enumerator Hot_pink
    0xFB56

enumerator Hunter_green
    0x32E7

enumerator Hunyadi_yellow
    0xE548

enumerator Ice_blue
    0x9FFF

enumerator Iceberg
    0x753A

enumerator Icterine
    0xFFAB

enumerator Illuminating_emerald
0x348E

enumerator Imperial_red
0xE947

enumerator Inchworm
0xB74B

enumerator Independence
0x4A8D

enumerator India_green
0x1441

enumerator Indian_red
0xCAEB

enumerator Indian_yellow
0xE54B

enumerator Indigo
0x4810

enumerator Indigo_dye
0x020D

enumerator International_klein_blue
0x1051

enumerator International_orange
0xFA80

enumerator International_orange_aerospace
0xFA80

enumerator International_orange_engineering
0xB8A1

enumerator International_orange_golden_gate_bridge
0xB9A5

enumerator Iris
0x5A99

enumerator Irresistible
    0xB22D

enumerator Isabelline
    0xF77D

enumerator Islamic_green
    0x0480

enumerator Italian_sky_blue
    0xB7FF

enumerator Ivory
    0xFFFD

enumerator Jade
    0x054D

enumerator Japanese_carmine
    0x9946

enumerator Japanese_violet
    0x598A

enumerator Jasmine
    0xF6EF

enumerator Jasper
    0xCAA8

enumerator Jazzberry_jam
    0xA06B

enumerator Jet
    0x31A6

enumerator Jonquil
    0xF643

enumerator Jordy_blue
    0x8DDD

enumerator June_bud
    0xBECB

enumerator Jungle_green
0x2D50

enumerator Kelly_green
0x4DC3

enumerator Keppel
0x3D73

enumerator Key_lime
0xE791

enumerator Khaki
0xC572

enumerator Khaki_web
0xC572

enumerator Khaki_x11_light_khaki
0xEF31

enumerator Kobe
0x8963

enumerator Kobi
0xE4F8

enumerator Kobicha
0x6A24

enumerator Kombu_green
0x3206

enumerator Ksu_purple
0x5151

enumerator Languid_lavender
0xD65B

enumerator Lapis_lazuli
0x2B13

enumerator Laser_lemon
0xFFEC

enumerator Laurel_green
0xADD3

enumerator Lava
0xC884

enumerator Lavender
0xB3FB

enumerator Lavender_blue
0xCE5F

enumerator Lavender_blush
0xFF7E

enumerator Lavender_floral
0xB3FB

enumerator Lavender_gray
0xC619

enumerator Lavender_pink
0xFD7A

enumerator Lavender_web
0xE73E

enumerator Lawn_green
0x7FC0

enumerator Lemon
0xFFA0

enumerator Lemon_chiffon
0xFFD9

enumerator Lemon_crayola
0xFFE0

enumerator Lemon_curry
0xCD04

enumerator Lemon_glacier
0xFFE0

enumerator Lemon_meringue
0xF757

enumerator Lemon_yellow
0xFF8A

enumerator Lemon_yellow_crayola
0xFFF3

enumerator Liberty
0x52D4

enumerator Licorice
0x1882

enumerator Light_blue
0xAEBC

enumerator Light_coral
0xEC10

enumerator Light_cornflower_blue
0x965C

enumerator Light_cyan
0xDFFF

enumerator Light_deep_pink
0xFAF9

enumerator Light_french_beige
0xC56F

enumerator Light_goldenrod_yellow
0xF7DA

enumerator Light_gray
0xD69A

enumerator Light_green
0x9772

enumerator Light_hot_pink
0xFD9B

enumerator Light_orange
: 0xFEB6

enumerator Light_periwinkle
: 0xC65B

enumerator Light_pink
: 0xFDB7

enumerator Light_red
: 0xFBEF

enumerator Light_salmon
: 0xFD0F

enumerator Light_sea_green
: 0x2595

enumerator Light_sky_blue
: 0x867E

enumerator Light_slate_gray
: 0x7453

enumerator Light_steel_blue
: 0xAE1B

enumerator Light_yellow
: 0xFFFB

enumerator Lilac
: 0xC518

enumerator Lilac_luster
: 0xACD5

enumerator Lime_color_wheel
: 0xBFE0

enumerator Lime_green
: 0x3666

enumerator Lime_web_x11_green
: 0x07E0

enumerator Lincoln_green
0x1AC1

enumerator Linen
0xF77C

enumerator Lion
0xBCCD

enumerator Liseran_purple
0xDB74

enumerator Little_boy_blue
0x6D1B

enumerator Liver
0x6A69

enumerator Liver_chestnut
0x93AA

enumerator Liver_dogs
0xB365

enumerator Liver_organ
0x6964

enumerator Livid
0x64D9

enumerator Lust
0xE104

enumerator Macaroni_and_cheese
0xFDF1

enumerator Madder
0xA004

enumerator Madder_lake
0xC9A7

enumerator Magenta
0xF81F

enumerator Magenta_additive_secondary
0xF81F

enumerator Magenta_crayola
0xF2B4

enumerator Magenta_dye
0xC90F

enumerator Magenta_haze
0x9A2E

enumerator Magenta_pantone
0xCA0F

enumerator Magenta_process
0xF812

enumerator Magenta_subtractive_primary
0xF812

enumerator Magic_mint
0xAF79

enumerator Magnolia
0xEF3A

enumerator Mahogany
0xBA00

enumerator Maize
0xFF4B

enumerator Maize_crayola
0xEE29

enumerator Majorelle_blue
0x629B

enumerator Malachite
0x0ECA

enumerator Manatee
0x94D5

enumerator Mandarin
0xF3C9

enumerator Mango
0xFDE0

enumerator Mango_tango
0xFC08

enumerator Mantis
0x760C

enumerator Mardi_gras
0x8810

enumerator Marengo
0x4ACC

enumerator Marigold
0xE504

enumerator Maroon
0x8000

enumerator Maroon_crayola
0xC109

enumerator Maroon_web
0x8000

enumerator Maroon_x11
0xA98C

enumerator Mauve
0xDD7F

enumerator Mauve_mallow
0xDD7F

enumerator Mauve_taupe
0x92ED

enumerator Mauvelous
0xECD5

enumerator Maximum_blue
    0x4D59

enumerator Maximum_blue_green
    0x35F7

enumerator Maximum_blue_purple
    0xAD5C

enumerator Maximum_green
    0x5C66

enumerator Maximum_green_yellow
    0xD72A

enumerator Maximum_purple
    0x71B0

enumerator Maximum_red
    0xD104

enumerator Maximum_red_purple
    0xA1CF

enumerator Maximum_yellow
    0xF7C7

enumerator Maximum_yellow_red
    0xEDC9

enumerator May_green
    0x4C88

enumerator Maya_blue
    0x761F

enumerator Medium_aquamarine
    0x66F5

enumerator Medium_blue
    0x0019

enumerator Medium_candy_apple_red
    0xD825

enumerator Medium_carmine
0xAA06

enumerator Medium_champagne
0xF735

enumerator Medium_gray
0xBDF7

enumerator Medium_orchid
0xBABA

enumerator Medium_purple
0x939B

enumerator Medium_sea_green
0x3D8E

enumerator Medium_slate_blue
0x7B5D

enumerator Medium_spring_green
0x07D3

enumerator Medium_turquoise
0x4E99

enumerator Medium_violet_red
0xC0B0

enumerator Mellow_apricot
0xF5AF

enumerator Mellow_yellow
0xF6EF

enumerator Melon
0xFDD5

enumerator Metallic_gold
0xD567

enumerator Metallic_seaweed
0x0BF1

enumerator Metallic_sunburst
0x9BE7

enumerator Mexican_pink
0xE00F

enumerator Middle_blue
0x7E9C

enumerator Middle_blue_green
0x8ED9

enumerator Middle_blue_purple
0x8B97

enumerator Middle_green
0x4C6B

enumerator Middle_green_yellow
0xADEC

enumerator Middle_grey
0x8C30

enumerator Middle_purple
0xD416

enumerator Middle_red
0xE46E

enumerator Middle_red_purple
0xA2AA

enumerator Middle_yellow
0xFF40

enumerator Middle_yellow_red
0xED8E

enumerator Midnight
0x712E

enumerator Midnight_blue
0x18CE

enumerator Midnight_green
0x026A

enumerator Midnight_green_eagle_green
0x024A

enumerator Mikado_yellow
0xFE01

enumerator Mimi_pink
0xFEDC

enumerator Mindaro
0xE7D1

enumerator Ming
0x3BAF

enumerator Minion_yellow
0xF6EA

enumerator Mint
0x4591

enumerator Mint_cream
0xF7FE

enumerator Mint_green
0x97D2

enumerator Misty_moss
0xBD8E

enumerator Misty_rose
0xFF1B

enumerator Mode_beige
0x9383

enumerator Moonstone
0x3D57

enumerator Morning_blue
0x8D13

enumerator Moss_green
    0x8CCB

enumerator Mountain_meadow
    0x35D1

enumerator Mountbatten_pink
    0x9BD1

enumerator Msu_green
    0x1A27

enumerator Mulberry
    0xC271

enumerator Mulberry_crayola
    0xC293

enumerator Mustard
    0xFECB

enumerator Myrtle_green
    0x33CE

enumerator Mystic
    0xD290

enumerator Mystic_maroon
    0xAA2F

enumerator Nadeshiko_pink
    0xF578

enumerator Naples_yellow
    0xF6CB

enumerator Navajo_white
    0xFEF5

enumerator Navy_blue
    0x0010

enumerator Navy_blue_crayola
    0x1BBA

enumerator Ndhu_green
0x12E6

enumerator Neon_blue
0x4A7F

enumerator Neon_carrot
0xFD08

enumerator Neon_fuchsia
0xFA0C

enumerator Neon_green
0x3FE2

enumerator New_york_pink
0xD40F

enumerator Nickel
0x73AE

enumerator Non_photo_blue
0xA6FD

enumerator Northwestern_purple
0x4950

enumerator Nyanza
0xE7FB

enumerator Ocean_blue
0x5216

enumerator Ocean_green
0x4DF2

enumerator Ochre
0xCBA4

enumerator Old_burgundy
0x4186

enumerator Old_gold
0xCDA7

enumerator Old_lace
0xFFBC

enumerator Old_lavender
0x7B4F

enumerator Old_mauve
0x6989

enumerator Old_rose
0xBC10

enumerator Old_silver
0x8430

enumerator Olive
0x8400

enumerator Olive_drab_3
0x6C64

enumerator Olive_drab_7
0x39A4

enumerator Olive_green
0xB58B

enumerator Olive_ral
0x39E7

enumerator Olivine
0x9DCE

enumerator Onyx
0x31C7

enumerator Opal
0xA617

enumerator Opera_mauve
0xB434

enumerator Orange
0xFBE0

enumerator Orange_color_wheel
0xFC00

enumerator Orange_crayola
0xFBA7

enumerator Orange_gs
0xE300

enumerator Orange_pantone
0xFAC0

enumerator Orange_peel
0xFCE0

enumerator Orange_red
0xFB44

enumerator Orange_red_crayola
0xFAA9

enumerator Orange_soda
0xF2C7

enumerator Orange_web
0xFD20

enumerator Orange_web_color
0xFD20

enumerator Orange_yellow
0xF5E4

enumerator Orange_yellow_crayola
0xF6AD

enumerator Orchid
0xDB9A

enumerator Orchid_crayola
0xDCFA

enumerator Orchid_pink
0xEDF9

enumerator Ou_crimson_red
0x80A3

enumerator Outer_space
0x4249

enumerator Outer_space_crayola
0x29C7

enumerator Outrageous_orange
0xFB69

enumerator Oxblood
0x4800

enumerator Oxford_blue
0x0109

enumerator Pacific_blue
0x1D58

enumerator Pakistan_green
0x0320

enumerator Palatinate
0x712D

enumerator Palatinate_purple
0x694C

enumerator Pale_aqua
0xBE9C

enumerator Pale_azure
0x869E

enumerator Pale_cerulean
0x9E1B

enumerator Pale_dogwood
0xEE78

enumerator Pale_green
0x97D2

enumerator Pale_pink
0xF659

enumerator Pale_purple
0xF73E

enumerator Pale_purple_pantone
0xF73E

enumerator Pale_silver
0xC5F7

enumerator Pale_spring_bud
0xEF57

enumerator Pansy_purple
0x78C9

enumerator Paolo_veronese_green
0x04CF

enumerator Papaya_whip
0xFF7A

enumerator Paradise_pink
0xE1EC

enumerator Parchment
0xEF5A

enumerator Paris_green
0x562F

enumerator Pastel_pink
0xDD34

enumerator Patriarch
0x8010

enumerator Paynes_grey
0x534F

enumerator Peach
0xFF36

enumerator Peach_crayola
    0xFE54

enumerator Peach_puff
    0xFED6

enumerator Pear
    0xCF06

enumerator Pearly_purple
    0xB354

enumerator Periwinkle
    0xCE5F

enumerator Periwinkle_crayola
    0xC67C

enumerator Permanent_geranium_lake
    0xD965

enumerator Persian_blue
    0x19D7

enumerator Persian_green
    0x0532

enumerator Persian_indigo
    0x308F

enumerator Persian_orange
    0xD48B

enumerator Persian_pink
    0xF3F7

enumerator Persian_plum
    0x70E3

enumerator Persian_red
    0xC9A6

enumerator Persian_rose
    0xF954

enumerator Persimmon
0xEAC0

enumerator Peru
0xCC28

enumerator Pewter_blue
0x8D56

enumerator Phlox
0xD81F

enumerator Phthalo_blue
0x0091

enumerator Phthalo_green
0x11A4

enumerator Picotee_blue
0x3150

enumerator Picton_blue
0x459C

enumerator Pictorial_carmine
0xC069

enumerator Piggy_pink
0xFEFC

enumerator Pine_green
0x03CD

enumerator Pine_tree
0x2984

enumerator Pink
0xFDF9

enumerator Pink_flamingo
0xFBBF

enumerator Pink_gs
0xF5F7

enumerator Pink_lace
0xFEFE

enumerator Pink_lavender
0xDD99

enumerator Pink_pantone
0xD252

enumerator Pink_sherbet
0xF474

enumerator Pistachio
0x962E

enumerator Platinum
0xE71B

enumerator Plum
0x8A30

enumerator Plum_crayola
0x818F

enumerator Plum_web
0xDD1B

enumerator Plump_purple
0x5A36

enumerator Polished_pine
0x5D32

enumerator Polynesian_blue
0x2272

enumerator Pomp_and_power
0x8311

enumerator Popstar
0xBA8C

enumerator Portland_orange
0xFAC7

enumerator Powder_blue
0xAEFC

enumerator Princeton_orange
0xEBE5

enumerator Process_yellow
0xFF60

enumerator Prune
0x70E3

enumerator Prussian_blue
0x018A

enumerator Psychedelic_purple
0xD81F

enumerator Puce
0xCC53

enumerator Pullman_brown_ups_brown
0x6203

enumerator Pumpkin
0xFBA3

enumerator Purple
0x8010

enumerator Purple_htmlcss_color
0x8010

enumerator Purple_mountain_majesty
0x93D6

enumerator Purple_munsell
0x9818

enumerator Purple_navy
0x4A90

enumerator Purple_pizzazz
0xFA7B

enumerator Purple_plum
0x9A96

enumerator Purple_web
0x8010

enumerator Purple_x11
0x991D

enumerator Purple_x11_color
0x991D

enumerator Purpureus
0x9A75

enumerator Queen_blue
0x4352

enumerator Queen_pink
0xE65A

enumerator Quick_silver
0xA534

enumerator Quinacridone_magenta
0x988A

enumerator Radical_red
0xF9AB

enumerator Raisin_black
0x2104

enumerator Rajah
0xFD4C

enumerator Raspberry
0xE06B

enumerator Raspberry_glace
0x92ED

enumerator Raspberry_rose
0xB22D

enumerator Raw_sienna
0xD44B

enumerator Raw_umber
0x8328

enumerator Razzle_dazzle_rose
0xF9B9

enumerator Razzmatazz
0xE12D

enumerator Razzmic_berry
0x8A70

enumerator Rebecca_purple
0x61B3

enumerator Red
0xF800

enumerator Red_brown
0xA145

enumerator Red_cmyk_pigment_red
0xE8E4

enumerator Red_crayola
0xE909

enumerator Red_munsell
0xE807

enumerator Red_ncs
0xC006

enumerator Red_orange
0xFAA9

enumerator Red_orange_color_wheel
0xFA20

enumerator Red_orange_crayola
0xFB44

enumerator Red_pantone
: 0xE947

enumerator Red_pigment
: 0xE8E4

enumerator Red_purple
: 0xE00F

enumerator Red_rgb
: 0xF800

enumerator Red_ryb
: 0xF942

enumerator Red_salsa
: 0xF9C9

enumerator Red_violet
: 0xC0B0

enumerator Red_violet_color_wheel
: 0x9168

enumerator Red_violet_crayola
: 0xBA31

enumerator Redwood
: 0xA2CA

enumerator Resolution_blue
: 0x0130

enumerator Rhythm
: 0x73B2

enumerator Rich_black
: 0x0208

enumerator Rich_black_fogra29
: 0x0062

enumerator Rich_black_fogra39
: 0x0000

enumerator Rifle_green
0x4267

enumerator Robin_egg_blue
0x0659

enumerator Rocket_metallic
0x8BF1

enumerator Rojo
0xE005

enumerator Rojo_spanish_red
0xA880

enumerator Roman_silver
0x8452

enumerator Rose
0xF80F

enumerator Rose_bonbon
0xF213

enumerator Rose_dust
0x9AED

enumerator Rose_ebony
0x6A49

enumerator Rose_madder
0xE127

enumerator Rose_pink
0xFB39

enumerator Rose_pompadour
0xEBD3

enumerator Rose_quartz
0xACD5

enumerator Rose_red
0xC0EA

enumerator Rose_taupe
0x92EB

enumerator Rose_vale
0xAA6A

enumerator Rosewood
0x6001

enumerator Rosso_corsa
0xD000

enumerator Rosy_brown
0xBC71

enumerator Royal_blue_dark
0x012C

enumerator Royal_blue_light
0x435B

enumerator Royal_blue_traditional
0x012C

enumerator Royal_blue_web_color
0x435B

enumerator Royal_purple
0x7A95

enumerator Royal_yellow
0xF6CB

enumerator Ruber
0xCA2E

enumerator Rubine_red
0xC80A

enumerator Ruby
0xD88C

enumerator Ruby_red
0x9884

enumerator Ruddy_blue
0x755B

enumerator Rufous
0xA0E1

enumerator Russet
0x8223

enumerator Russian_green
0x6C8D

enumerator Russian_violet
0x30C9

enumerator Rust
0xB202

enumerator Rusty_red
0xD968

enumerator Sacramento_state_green
0x01C5

enumerator Saddle_brown
0x8A22

enumerator Safety_orange
0xFBC0

enumerator Safety_orange_blaze_orange
0xFB20

enumerator Safety_yellow
0xEE80

enumerator Saffron
0xF606

enumerator Sage
0xBDB1

enumerator Salmon
0xF40E

enumerator Salmon_pink
0xFC94

enumerator Sand
0xC590

enumerator Sand_dune
0x9383

enumerator Sandy_brown
0xF52C

enumerator Sap_green
0x53E5

enumerator Sapphire
0x092D

enumerator Sapphire_blue
0x0334

enumerator Sapphire_crayola
0x0334

enumerator Satin_sheen_gold
0xCD06

enumerator Savoy_blue
0x4B19

enumerator Scarlet
0xF920

enumerator Schauss_pink
0xFC95

enumerator School_bus_yellow
0xFEA0

enumerator Screamin_green
0x67EC

enumerator Sea_green
0x344B

enumerator Sea_green_crayola
0x07F9

enumerator Seal_brown
0x5921

enumerator Seashell
0xFFBD

enumerator Selective_yellow
0xFDC0

enumerator Sepia
0x7202

enumerator Sgbus_green
0x56E6

enumerator Shadow
0x8BCB

enumerator Shadow_blue
0x7454

enumerator Shamrock_green
0x04EC

enumerator Sheen_green
0x8E80

enumerator Shimmering_blush
0xD432

enumerator Shiny_shamrock
0x652F

enumerator Shocking_pink
0xF897

enumerator Shocking_pink_crayola
0xFB7F

enumerator Shocking_pink_crayola_formerly_known_as_ultra_pink
0xFB7F

enumerator Sienna
0x8963

enumerator Silver
0xBDF7

enumerator Silver_chalice
0xAD55

enumerator Silver_crayola
0xC5F7

enumerator Silver_lake_blue
0x5C57

enumerator Silver_metallic
0xAD55

enumerator Silver_pink
0xC575

enumerator Silver_sand
0xBE18

enumerator Sinopia
0xCA01

enumerator Sizzling_red
0xF9CA

enumerator Sizzling_sunrise
0xFEC0

enumerator Skobeloff
0x03CE

enumerator Sky_blue
0x867D

enumerator Sky_blue_crayola
0x86DD

enumerator Sky_magenta
0xCB95

enumerator Slate_blue
0x6AD9

enumerator Slate_gray
0x7412

enumerator Slimy_green
0x2CA3

enumerator Smitten
0xC210

enumerator Smokey_topaz
0x8142

enumerator Smoky_black
0x1061

enumerator Snow
0xFFDE

enumerator Solid_pink
0x89C8

enumerator Sonic_silver
0x73AE

enumerator Space_cadet
0x214A

enumerator Spanish_bistre
0x83A6

enumerator Spanish_blue
0x0397

enumerator Spanish_carmine
0xC809

enumerator Spanish_gray
0x94D2

enumerator Spanish_green
0x048A

enumerator Spanish_orange
0xE300

enumerator Spanish_pink
0xF5F7

enumerator Spanish_red
0xE005

enumerator Spanish_sky_blue
0x07FF

enumerator Spanish_violet
0x4950

enumerator Spanish_viridian
0x03EB

enumerator Spring_bud
0xA7C0

enumerator Spring_frost
0x87E5

enumerator Spring_green
0x07EF

enumerator Spring_green_crayola
0xEF57

enumerator St_patricks_blue
0x214F

enumerator Star_command_blue
0x03D6

enumerator Steel_blue
0x4C16

enumerator Steel_pink
0xC9B9

enumerator Steel_teal
0x6451

enumerator Stil_de_grain_yellow
0xF6CB

enumerator Stone_gray
0x9470

enumerator Straw
0xE6CD

enumerator Strawberry
0xF28A

enumerator Strawberry_blonde
0xFC8C

enumerator Sugar_plum
0x926E

enumerator Sunglow
0xFE46

enumerator Sunray
0xE54B

enumerator Sunset
0xF6B4

enumerator Super_pink
0xCB55

enumerator Sweet_brown
0xA1C6

enumerator Syracuse_orange
0xD220

enumerator Tan
0xD591

enumerator Tan_crayola
0xD4CD

enumerator Tang_blue
0x02D9

enumerator Tangerine
0xEC20

enumerator Tango_pink
0xE38F

enumerator Tart_orange
0xFA69

enumerator Taupe
0x49E6

enumerator Taupe_gray
0x8C31

enumerator Tea_green
0xCF77

enumerator Tea_rose
0xF40F

enumerator Tea_rose_red
0xF618

enumerator Teal
0x0410

enumerator Teal_blue
0x3BB1

enumerator Telemagenta
0xC9AE

enumerator Tenne
0xCAA0

enumerator Terra_cotta
0xDB8B

enumerator Thistle
0xD5FA

enumerator Thulian_pink
0xDB74

enumerator Tickle_me_pink
0xFC55

enumerator Tiffany_blue
0x86B9

enumerator Tigers_eye
0xB343

enumerator Timberwolf
0xDEBA

enumerator Titanium_yellow
0xEF20

enumerator Tomato
0xFB09

enumerator Tropical_rainforest
0x03AB

enumerator True_blue
0x2B58

enumerator Trypan_blue
0x1836

enumerator Tufts_blue
0x447B

enumerator Tumbleweed
0xDD51

enumerator Turkey_red
0xA880

enumerator Turquoise
0x46F9

enumerator Turquoise_blue
0x07FD

enumerator Turquoise_green
0x9EB6

enumerator Turtle_green
    0x8CCB

enumerator Tuscan
    0xF6B4

enumerator Tuscan_brown
    0x6A67

enumerator Tuscan_red
    0x7A49

enumerator Tuscan_tan
    0xA3CB

enumerator Tuscany
    0xBCD3

enumerator Twilight_lavender
    0x8A4D

enumerator Tyrian_purple
    0x6007

enumerator Ua_blue
    0x01B5

enumerator Ua_red
    0xD009

enumerator Ultra_pink
    0xFB7F

enumerator Ultra_red
    0xFB70

enumerator Ultra_violet
    0x62B2

enumerator Ultramarine
    0x401F

enumerator Ultramarine_blue
    0x433E

enumerator Umber
0x6289

enumerator Unbleached_silk
0xFEF9

enumerator United_nations_blue
0x4C9B

enumerator University_of_pennsylvania_blue
0x010B

enumerator University_of_pennsylvania_red
0x9800

enumerator Unmellow_yellow
0xFFEC

enumerator Up_forest_green
0x0224

enumerator Up_maroon
0x7882

enumerator Upsdell_red
0xA905

enumerator Uranian_blue
0xAEDE

enumerator Usafa_blue
0x0292

enumerator Ut_orange
0xFC00

enumerator Van_dyke_brown
0x6205

enumerator Vanilla
0xF735

enumerator Vanilla_ice
0xF475

enumerator Vegas_gold
    0xC58B

enumerator Venetian_red
    0xC043

enumerator Verdigris
    0x4595

enumerator Vermilion
    0xE206

enumerator Veronica
    0x991D

enumerator Violet
    0x781F

enumerator Violet_blue
    0x3256

enumerator Violet_blue_crayola
    0x7378

enumerator Violet_color_wheel
    0x781F

enumerator Violet_crayola
    0x91EF

enumerator Violet_gs
    0x4950

enumerator Violet_jtc
    0x598A

enumerator Violet_red
    0xF2B2

enumerator Violet_ryb
    0x8015

enumerator Violet_web
    0xEC1D

enumerator Violet_web_color
0xEC1D

enumerator Viridian
0x440D

enumerator Viridian_green
0x04B2

enumerator Vista_blue
0x7CFA

enumerator Vivid_burgundy
0x98E6

enumerator Vivid_sky_blue
0x065F

enumerator Vivid_tangerine
0xFD11

enumerator Vivid_violet
0x981F

enumerator Volt
0xCFE0

enumerator Walnut_brown
0x5A89

enumerator Warm_black
0x0208

enumerator Wenge
0x62AA

enumerator Wheat
0xF6F6

enumerator White
0xFFFF

enumerator White_smoke
0xF7BE

enumerator Wild_blue_yonder
    0xA579

enumerator Wild_orchid
    0xD394

enumerator Wild_strawberry
    0xFA34

enumerator Wild_watermelon
    0xFB70

enumerator Windsor_tan
    0xA2A0

enumerator Wine
    0x7187

enumerator Wine_dregs
    0x6989

enumerator Winter_sky
    0xF80F

enumerator Wintergreen_dream
    0x544F

enumerator Wisteria
    0xC51B

enumerator Wood_brown
    0xBCCD

enumerator Xanadu
    0x742F

enumerator Xanthic
    0xEF61

enumerator Xanthous
    0xED86

enumerator Yale_blue
    0x01AD

enumerator Yale_blue_site_blue
: 0x01AD

enumerator Yellow
: 0xFFE0

enumerator Yellow_crayola
: 0xFF30

enumerator Yellow_green
: 0x9E66

enumerator Yellow_green_color_wheel
: 0x3583

enumerator Yellow_green_crayola
: 0xC710

enumerator Yellow_munsell
: 0xEE40

enumerator Yellow_ncs
: 0xFE80

enumerator Yellow_orange
: 0xFD68

enumerator Yellow_orange_color_wheel
: 0xFCA1

enumerator Yellow_pantone
: 0xFEE0

enumerator Yellow_process
: 0xFF60

enumerator Yellow_rgb_x11_yellow
: 0xFFE0

enumerator Yellow_ryb
: 0xFFE6

enumerator Yellow_sunshine
: 0xFFA0

enumerator Yinmn_blue
: 0x3292

enumerator Zaffre
: 0x00B4

enumerator Zomp
: 0x3D31

6.14 Налаштування бібліотеки

Бібліотека lilka постачається з рядом вбудованих можливостей, які за замовчуванням увімкнені.

Можливо, вам не подобаються деякі налаштування, які вибрані за замовчуванням.

Наприклад, вас дратує звук, який видає Лілка при увімкненні. Або ж ви збираєте Лілку на макетній платі і ваш дисплей не здатний працювати на швидкості 80 МГц через високий рівень шумів та ємкостей, викликаних макетною платою та перемичками.

Завдяки PlatformIO ви можете змінити налаштування, відредагувавши файл platformio.ini. Для цього є опція build_flags.

Наприклад, для того, щоб вимкнути всі звуки, ви можете відредагувати файл firmware/main/platformio.ini та додати наступний рядок:

```
[env:v2]
platform = espressif32
board = lilka_v2
framework = arduino
lib_deps =
    lilka
build_flags = -D LILKA_NO_BUZZER_HELLO -D LILKA_NO_AUDIO_HELLO
```

Після цього ви повинні перекомпілювати вашу прошивку.

6.14.1 Перелік опцій

LILKA_BREADBOARD
: Використовуйте цю опцію, якщо ви збираєте Лілку на макетній платі або на іншій платі, яка має високий рівень шумів та ємкостей.

  Вона зменшує швидкість SPI-шини з 80 МГц до 40 МГц.

LILKA_NO_BUZZER_HELLO
: Встановіть цю опцію, щоб вимкнути звук, який Лілка відтворює через п'єзоелектричний динамік при увімкненні.

LILKA_NO_AUDIO_HELLO
: Встановіть цю опцію, щоб вимкнути звук, який Лілка відтворює через I2S-модуль при увімкненні.

Розділ 7

Поширені запитання

7.1 Загальна інформація

Зміст

7.1.1 Навіщо мені Лілка?

Лілка - це маленький, легкий і мобільний пристрій, який може бути використаний для різних цілей, таких як навчання, розробка програмного забезпечення, робототехніка, IoT, медіацентр, проста ігрова консоль і багато іншого.

Вона відносно проста і дешева, а також використовує доволі потужний мікроконтролер ESP32-S3-WROOM-1-N16R8, який має вбудований Wi-Fi та Bluetooth, а також 16 МБ зовнішньої RAM.

Основна мета Лілки - це бути дешевою, легкою в навчанні та сумісною з різними додатками, які були розроблені для неї іншими користувачами.

Якщо ви шукаєте потужну ігрову консоль, то Лілка, можливо, не найкращий вибір для вас. (Хоча на ній можна без проблем грати в досить популярні старі ігри, такі як Doom чи Wolfenstein, а також запускати на ній емулятори NES, SNES та інших консолей.)

7.1.2 Я хочу більший дисплей!

Лілка - це бюджетний пристрій, тому вона має досить маленький дисплей.

Крім того, кожен дисплей має власний контролер, який відповідає за відображення зображення. Це означає, що ви не зможете просто підключити більший дисплей до Лілки. Таких контролерів є багато і вони не сумісні між собою, а наша команда не має можливості тестувати та підтримувати кожен з них.

Однак, якщо ви знайдете дисплей з таким же контролером (і таким самим розташуванням контактів), як у Лілки, ви можете спробувати підключити його до Лілки, але це буде на ваш власний ризик.

Але не забувайте, що мета Лілки - це не лише бути маленьким, легким і мобільним пристроєм, а й бути сумісною з різними додатками, які були розроблені для неї іншими користувачами. Якщо ви підключите більший дисплей, деякі додатки можуть відображатися некоректно або взагалі не працювати.

Якщо вам потрібен більший дисплей, можливо, вам варто розглянути інші пристрої.

7.1.3 Чому Лілка називається Лілкою?

Я назвав цей пристрій на честь однієї з моїх кішок, яку звати Ліла, яку, в свою чергу, я назвав на честь Ліли з Футурами. Цей пристрій досить маленький та простий і асоціюється мені з англійським словом «little», тобто «маленький». Звідси і виникла ідея для назви. :)

Ліла

Льоля та Ліла

Крім того, Лілка - це українське жіноче ім'я, яке походить від слова «лілея». Лілея - це квітка, яка символізує чистоту, вірність, вічність, а також відновлення і відродження.

7.2 Апаратна частина

Зміст

7.2.1 Я не можу завантажити прошивку на свій пристрій. Що робити?

Переконайтеся, що ви використовуєте саме так званий «data» кабель, а не «power» кабель. «Power» кабель призначений лише для заряджання пристрою, а «data» кабель - і для передачі даних, і для заряджання.

> Попередження: Зверніть увагу: Лілка не підтримує USB 3.0, тому ви не зможете прошивати її через кабель Type C - Type C. Використовуйте кабель Type C - Type A.

7.2.2 Що це за цифри в кружечках на платі Лілки?

Цифри в кружечках на платі Лілки - це номери GPIO-пінів мікроконтролера, які позначають, до якого GPIO підключений той чи інший компонент Лілки.

> Попередження: Не плутайте номери GPIO з номерами пінів! Наприклад, GPIO8 - це пін 12, а не 8. Радимо вам завжди користуватися номерами GPIO, а не номерами пінів - як в коді, так і в документації.
>
> Детальніше про піни ESP32-S3-WROOM-1 можна дізнатися в офіційній документації ESP32-S3-WROOM-1.

Якщо ви хочете підключити до Лілки якийсь сторонній модуль, ми рекомендуємо використовувати для цього Роз'єм розширення, де жоден пін не зайнятий внутрішніми компонентами Лілки.

Але оскільки Лілка - це відкрита DIY-консоль, ніхто не забороняє вам використовувати будь-які піни з нестандартною метою. Проте майте на увазі, що в такому випадку Бібліотека lilka не працюватиме належним чином, оскільки вона очікує, що до цих пінів будуть підключені конкретні відомі їй компоненти. Тому вам доведеться самостійно писати власний код для роботи з цими пінами.

7.2.3 Моя програма взаємодіє одночасно з SD-картою та дисплеєм, але вона працює дуже повільно!

Для роботи з SD-картою та дисплеєм Лілка використовує одну шину SPI.

Це означає, що в будь-який момент часу шина може бути зайнятою або SD-картою, або дисплеєм, і спроба взаємодії з одним пристроєм, коли інший пристрій використовує шину, призведе до затримок і, як наслідок, до зниження частоти кадрів та швидкості читання/запису файлів.

Для цього є декілька причин:

- Це дозволяє зменшити кількість пінів, задіяних для підключення SD-карти та дисплею.
- ESP32-S3 має всього дві шини SPI, тому використання однієї шини для обох пристроїв дозволяє звільнити іншу шину для потреб користувача (наприклад, для підключення зовнішньої камери чи акселерометра).

Щоб уникнути цих затримок, потрібно уникати роботи з SD-картою та дисплеєм одночасно. Найпростіший спосіб - це читати дані з SD-карти в пам'ять PSRAM (в ESP32-S3 її достатньо багато - аж 8 МБ), і надалі працювати з цими даними, не взаємодіючи з SD-картою.

Враховуючи, що (станом на момент написання цього документу) SD-карта працює на частоті 25 МГц, читання файлу розміром 1 МБ з SD-карти за ідеальних умов займе менше однієї секунди!

Тому, якщо ви пишете гру, яка завантажує текстури чи звуки з SD-карти, просто завантажте їх у пам'ять PSRAM один раз при запуску гри і надалі працюйте з ними вже з пам'яті, не взаємодіючи з SD-картою.

7.2.4 Чому ви не вивели *GPIO35*, *GPIO36* та *GPIO37* на роз'єм розширення?

Ці піни можна використовувати лише в деяких (менш потужних) варіантах ESP32-S3-WROOM.

В ESP32-S3-WROOM-N16R8, який використовується в Лілці, ці піни використовуються для внутрішіх потреб (а точніше - для спілкування мікроконтролера ESP32-S3 з пам'яттю PSRAM).

Тому використовувати їх для будь-яких інших цілей заборонено.

7.2.5 Чому ви не вивели *GPIO45* на роз'єм розширення?

GPIO45 - це один з так званих «strapping» пінів, які використовуються для вибору режиму завантаження мікроконтролера та інших налаштувань.

Використання цього піна небезпечне, оскільки він встановлює режим напруги SPI-шини. Тому ми вирішили не виводити його на роз'єм розширення, щоб уникнути можливих проблем та потенційних пошкоджень пристрою.

7.3 Програмування

Зміст

7.3.1 Що таке «Arduino», «ESP-IDF» і ця ваша «бібліотека lilka»?

- Arduino - це фреймворк, яка дозволяє швидко і просто створювати програми для мікроконтролерів.

  Основна перевага Arduino - це стандартизований і простий інтерфейс програмування, який дозволяє швидко створювати програми для мікроконтролерів. Неважливо, чи ви пишете код для ESP32, чи для STM32, чи для ATmega: ви можете використовувати однаковий інтерфейс програмування, однакові функції та навіть однакові бібліотеки.

  Тобто Arduino - це своєрідна абстракція, яка дозволяє вам писати код для мікроконтролерів, не замислюючись про те, який саме мікроконтролер ви використовуєте.

- ESP-IDF - це офіційний фреймворк для програмування мікроконтролерів сімейства ESP32 від компанії Espressif.

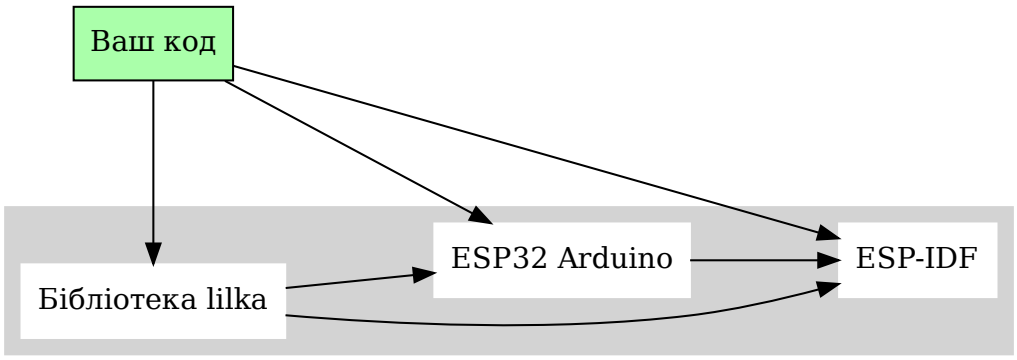
  ESP-IDF - це більш низькорівневий фреймворк, який дозволяє вам працювати з ESP32 на більш низькому рівні, ніж Arduino. Код, написаний на ESP-IDF, може бути більш швидким і ефективним, ніж код, написаний на Arduino, але він також буде складнішим і працюватиме тільки на мікроконтролерах сімействі ESP32.

- Бібліотека lilka - це бібліотека, яку ми написали для роботи з ESP32-S3 на платформі Arduino.

  Це - спроба поєднати простоту Arduino з ефективністю ESP-IDF саме для написання програм для Лілки, а також надати додаткові функції, які допоможуть вам створити програми для Лілки швидше і простіше.

Але не лякайтесь: ви можете одночасно використовувати і Arduino, і ESP-IDF, і бібліотеку lilka!

Річ у тім, що бібліотека lilka написана на основі ESP32-Arduino, а ESP32-Arduino, в свою чергу, використовує ESP-IDF. Це означає, що ви можете використовувати в своєму коді і функції ESP-IDF, і функції Arduino, і функції бібліотеки lilka.



7.3.2 Що таке «паніка»?

«Паніка» («panic») - це загальна назва критичних ситуацій, які виникають, коли програма не може продовжувати свою роботу через критичну помилку або через потенційну можливість пошкодження даних.

Класична паніка - це дереференсія нульового вказівника:

```
int *p = NULL;
*p = 42; // Паніка!
```

Проте паніка може бути викликана іншими причинами, такими як переповнення стеку, помилки вводу-виводу, помилки пам'яті, помилки в програмному коді, помилки в апаратурі, і так далі.

В більшості випадків при паніці ESP32 виводить в консоль не лише повідомлення про помилку, але і стек викликів, який привів до паніки.

Цей стек можна розкодувати, щоб побачити, які функції були викликані перед панікою і в якому порядку.

В KeiraOS для цього є «команда» decode_backtrace, який використовує утиліту addr2line з комплекту ESP-IDF для розкодування стеку викликів. Ось як його використовувати:

```
pio run -e v2 -t decode_backtrace -a '0x42013efa:0x3fcbbc10 0x4200894d:0x3fcbbcd0␣
↪0x4201962e:0x3fcbbd10 0x4201987b:0x3fcbbd30 0x4202724c:0x3fcbbd60'
```

Це виведе розкодовані функції та рядки, які викликалися перед панікою, від «найглибшої» до «найповерхневішої». Тобто перший рядок - це момент, де відбулася паніка.

Детальнішу інформацію про паніки можна знайти в документації ESP-IDF.

7.3.3 Я отримую паніку Stack canary watchpoint triggered

В ESP32 є спеціальний механізм, який дозволяє виявити переповнення стеку. Цей механізм називається «стекова канарка» (stack canary). Він слідкує за переповненням стеку задачі.

Оскільки в FreeRTOS кожна задача при створенні отримує свою власну область стеку, яка визначається параметром stack_depth функцій xTaskCreate..., найчастіше ця помилка виникає через недостатньо велику область стеку вашої задачі. Ви можете збільшити розмір її стеку, визначивши більший розмір стеку при виклику xTaskCreate....

> Попередження: Збільшення розміру стеку означає, що ваша програма буде використовувати більше пам'яті.
>
> Лілка використовує мікроконтролер ESP32-S3, який має всього декілька сотень кілобайт внутрішньої оперативної пам'яті (RAM), але крім неї вона має зовнішню оперативну пам'ять, яка називається PSRAM.
>
> Стек задачі зберігається у внутрішній RAM.
>
> Якщо вам потрібно зберігати великі обсяги даних в пам'яті, то можливо замість збільшення стеку вам краще варто використовувати динамічне виділення пам'яті в «купі», використовуючи функцію ps_malloc, яка виділяє пам'ять у PSRAM (якої в Лілці аж 8 МБ).
>
> PSRAM дещо повільніша за внутрішню RAM, але вона дозволяє зберігати в десятки разів більше даних. В більшості випадків різниця в швидкості взагалі не буде помітною.

7.3.4 Я отримую паніку IllegalInstruction

Ця помилка найчастіше трапляється, коли задача FreeRTOS, створена за допомогою одної з функцій xTaskCreate..., доходить до кінця (або робить return) без виклику vTaskDelete(NULL).

Переконайтеся, що ваша задача завершується викликом vTaskDelete(NULL): іншими словами, викликайте vTaskDelete(NULL) замість return.

7.3.5 Я отримую паніку Task watchdog got triggered

Ця помилка виникає, коли задача не викликає vTaskDelay або vTaskDelayUntil протягом певного часу.

В ESP32 є механізм «сторожового таймера» (watchdog timer), який слідкує за тим, щоб задачі виконувалися вчасно. Якщо задача не викликає vTaskDelay або vTaskDelayUntil протягом певного часу (зазвичай 10 секунд), ви отримаєте помилку Task watchdog got triggered. Це робиться для того, щоб уникнути «зависання» задачі, яка не виконується.

Переконайтеся, що ваша задача час від часу викликає taskYIELD() чи vTaskDelay(1).

Якщо це неможливо - наприклад, якщо ви використовуєте сторонню бібліотеку, яка робить довгі і складні обчислення - ви можете збільшити таймаут сторожового таймера, викликавши esp_task_wdt_init з більшим значенням таймауту.

Розділ 8

Словник термінів

Прошивка
: Скомпільована програма, яка виконується на мікроконтролері.

  Переважно є бінарним файлом з розширенням .bin, який записується в пам'ять мікроконтролера, але також може бути завантажена в мікроконтролер WiFi, SD-карту тощо.

  Компіляція та завантаження прошивки в мікроконтролер зазвичай виконується за допомогою спеціальних програм, наприклад, PlatformIO.

Прошивання
: Процес запису прошивки в пам'ять мікроконтролера. Переважно здійснюється через такі інтерфейси, як USB або WiFi.

OTA
: «Over-The-Air», тобто «через мережу»: прошивання без кабельного підключення до комп'ютера. Процес прошивання через мережу або з SD-карти.

PlatformIO
: Крос-платформене середовище для розробки програмного забезпечення для вбудованих систем, яке базується на відкритому коді та підтримується спільнотою.

  Включає в себе відладчик, засоби для роботи з прошивками, бібліотеками, платами тощо.

Visual Studio Code
: Відкритий та безкоштовний крос-платформений редактор коду.

  Має велику кількість розширень, які дозволяють розширити його функціонал, зокрема підтримку PlatformIO.

Git
: Система керування версіями файлів та спільної роботи над ними.

  Дозволяє зберігати кілька версій файлів, відслідковувати зміни, відновлювати втрачені версії тощо.

Весь код Лілки та Doom, всі креслення та навіть вся ця документація зберігаються саме в Git в репозиторії на GitHub і доступні для всіх бажаючих абсолютно безкоштовно.

Lua
: Легка скриптова мова програмування, яка використовується для написання скриптів та програм для вбудованих систем.

  Вона чудово підходить для написання ігор!

PCB
: «Printed Circuit Board», тобто «друкована плата». Плата, на якій розміщені електронні компоненти.

THT
: «Through-Hole Technology», тобто «технологія з монтажем через отвори». Технологія монтажу електронних компонентів на плату через отвори.

SMD
: «Surface-Mount Device», тобто «прилади з монтажем на поверхню». Технологія монтажу електронних компонентів на поверхню плати.

ЦАП
: Цифро-аналоговий перетворювач (DAC). Електронний пристрій, який перетворює цифрові сигнали в аналогові. Найчастіше використовується для відтворення звуку.

DAC
: Digital-to-Analog Converter, тобто «цифро-аналоговий перетворювач».

  Див. ЦАП.

Розділ 9

Спільнота

9.1 Rustilka: Rust для Лілки

Автор: imbir

Rustilka - це мініпроєкт підтримки мови Rust для Лілки, який має таку саму мету, як і сама консоль - «полегшити» розробку.

Також цей проєкт чисто технічно незалежний від оригінального SDK: при наявності можливостей ми могли б мати IDF версію, але, на думку автора Rustilka, це не має сенсу.

Хочете писати код для Лілки на Rust? Тоді вам сюди!

9.2 MeowUI: Альтернативний UI-фреймворк для Лілки

Автор: kolodieiev

MeowUI - набір бібліотек, що спрощують створення графічного інтерфейсу користувача для електронних пристроїв на базі esp32 та esp32s3.

З самого початку даний проєкт створювався, як шаблон, для швидкої побудови багаторівневого графічного інтерфейсу в стилі “Quadratisch. Praktisch. Gut”. Мета - управління різноманітними датчиками за допомогою мікроконтролера ESP32.

Після знайомства з проєктом «Лілка», автором MeowUI було прийнято рішення про розширення та доопрацювання функціоналу MeowUI.

Детальніше про проєкт можна дізнатися тут.

9.3 Спілкування

- Весь код проєкту доступний на GitHub.
- Приєднуйтесь до нашого Discord-сервера!

Розділ 10

Indices and tables

- genindex

Індекс

I

L

M

N

O

P

Q

R

S

T

U

V

W