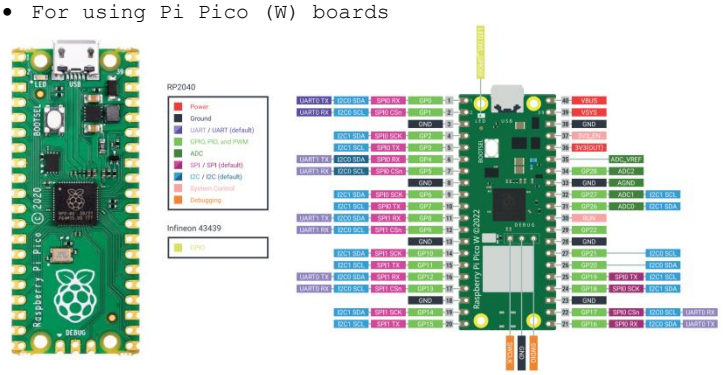
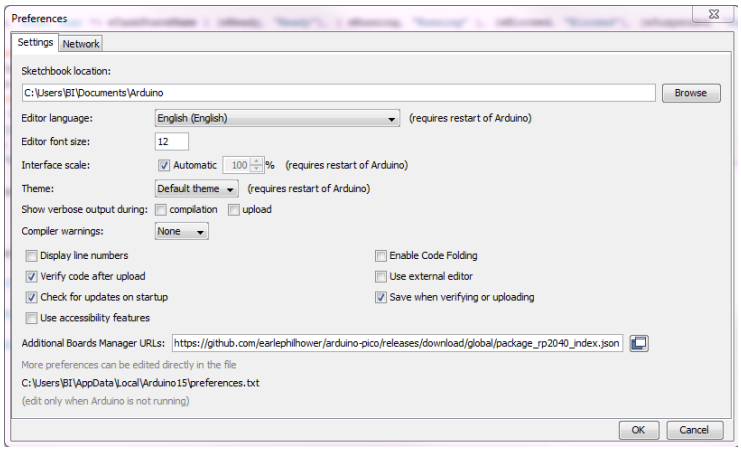
Arduino Pi Pico (W) boards

- For using Pi Pico (W) boards

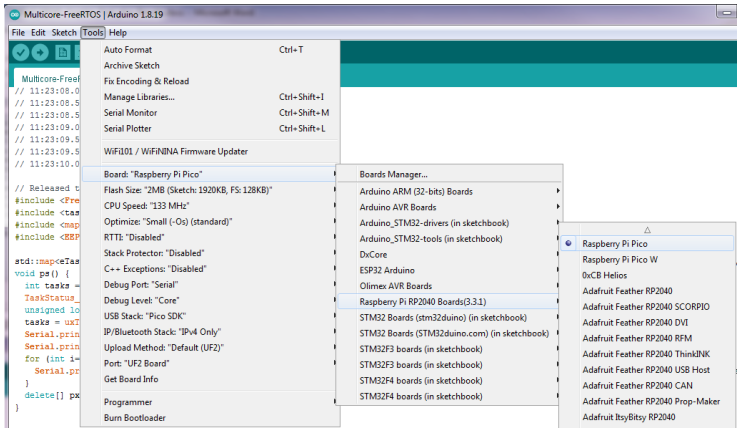


- In Preferences add URL: https://github.com/earlephilhower/arduino-pico/releases/download/global/package_rp2040_index.json



- Install Pi Pico / RP2040 in board manager

- Install "Raspberry Pi Pico" or "Raspberry Pi Pico W" board



- Connect the board to Windows PC while BOOTSEL button is pushed – "RPI-RP2" mass storage device should be appeared
- After uploading the sketch "Pico" or "Pico W" device will be appeared in "Device Manager"
- Update its device driver using Atmel USB to serial INF file changing [DeviceList.*] sections to: %PI_CDC_PICO%=DriverInstall, USB\VID_2E8A&PID_000A&REV_0100 or %PI_CDC_PICO%=DriverInstall, USB\VID_2E8A&PID_F00A&REV_0100
- Change [Strings] sections also to appropriate once

Multicore version of "Hello World and Blinking LED" common test for Pi Pico

- Open from File -> Examples -> (Examples for Paspberry Pi Pico) -> FreeRTOS -> Milticore FreeRTOS sketch and safe it in your Arduino sketch folder:

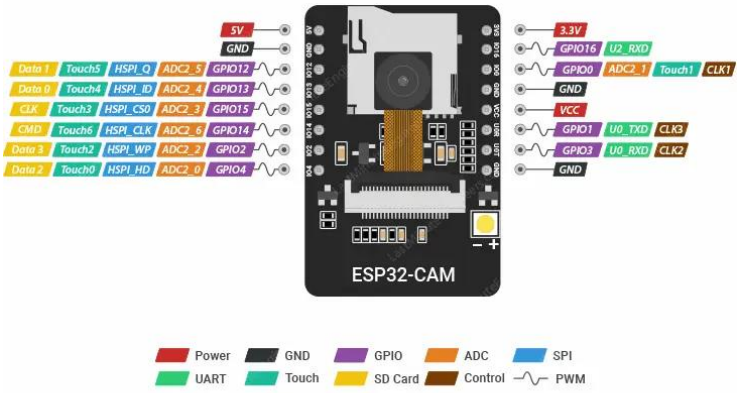
```
#include <FreeRTOS.h>
#include <task.h>
#include <map>
#include <EEPROM.h>
std::map<eTaskState, const char *> eTaskStateName {
{eReady, "Ready"}, { eRunning, "Running" }, {eBlocked,
"Blocked"}, {eSuspended, "Suspended"}, {eDeleted,
"Deleted"} };
void ps() {
  int tasks = uxTaskGetNumberOfTasks();
  TaskStatus_t *pxTaskStatusArray = new
TaskStatus_t[tasks];
  unsigned long runtime;
  tasks = uxTaskGetSystemState( pxTaskStatusArray, tasks,
&runtime );
  Serial.printf("# Tasks: %d\r\n", tasks);
  Serial.println("ID, NAME, STATE, PRIO, CYCLES");
  for (int i=0; i < tasks; i++) {
    Serial.printf("%d: %-16s %-10s %d %lu\r\n", i,
pxTaskStatusArray[i].pcTaskName,
eTaskStateName[pxTaskStatusArray[i].eCurrentState],
(int)pxTaskStatusArray[i].uxCurrentPriority,
pxTaskStatusArray[i].ulRunTimeCounter);
  }
  delete[] pxTaskStatusArray;
}
void blink(void *param) {
  (void) param;
  pinMode(LED_BUILTIN, OUTPUT);
  while (true) {
    digitalWrite(LED_BUILTIN, LOW);
    delay(750);
    digitalWrite(LED_BUILTIN, HIGH);
    delay(250);
  }
}
void setup() {
  Serial.begin(115200);
  xTaskCreate(blink, "BLINK", 128, nullptr, 1, nullptr);
  delay(5000);
}
volatile int val= 0;
void loop() {
  Serial.printf("C0: Blue leader standing by...\r\n");
  ps();
  Serial.printf("val: %d\r\n", val);
  delay(1000);
}
// Running on core1
void setup1() {
  delay(5000);
  Serial.printf("C1: Red leader standing by...\r\n");
}
void loop1() {
  static int x = 0;
  Serial.printf("C1: Stay on target...\r\n");
  val++;
  if (++x < 10) {
    EEPROM.begin(512);
    EEPROM.write(0,x);
    EEPROM.commit();
  }
  delay(1000);
}
```

- It demonstrates a simple use of the setup1()/loop1() functions for a multiprocessor run and following will be printed on the serial port while LED is blinking:

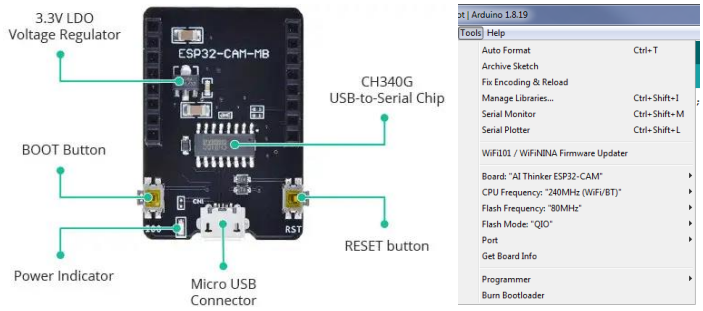
```
C1: Stay on target...
C0: Blue leader standing by...
# Tasks: 9
ID, NAME, STATE, PRIO, CYCLES
0: CORE0            Running    4 191473164
1: IDLE1            Running    0 3622023404
2: IDLE0            Ready      0 3371562651
3: BLINK            Blocked    1 5381238
4: CORE1            Blocked    4 103437988
5: USB              Blocked    6 3826967365
6: Tmr Svc          Blocked    2 21071
7: IdleCore1        Suspended  7 17213
8: IdleCore0        Suspended  7 88986822
val: 683
```

ESP32-CAM

- Getting Started With ESP32-CAM
- All examples work with ESP32 Espressif System 2.0.9



- Using ESP32-CAM-MB module makes programming easy



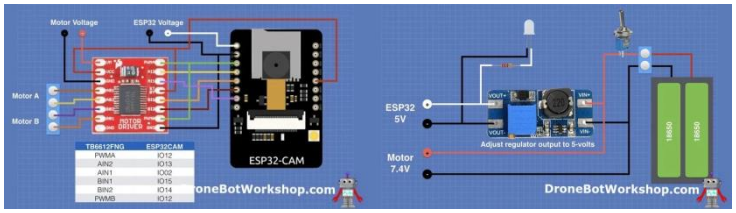
Wifi Camera Robot Car

- DIY ESP32 Camera Motor Shield - Wifi Camera Robot Car

https://www.olimex.com/Products/IoT/ESP32/ESP32-CAM/
https://www.instructables.com/DIY-ESP32-Camera-Motor-Shield-Wifi-Camera-Robot-Ca/,
https://dronebotworkshop.com/esp32-cam-intro/
https://randomnerdtutorials.com/esp32-cam-video-streaming-web-server-camera-home-assistant/
https://dronebotworkshop.com/esp32cam-robot-car/

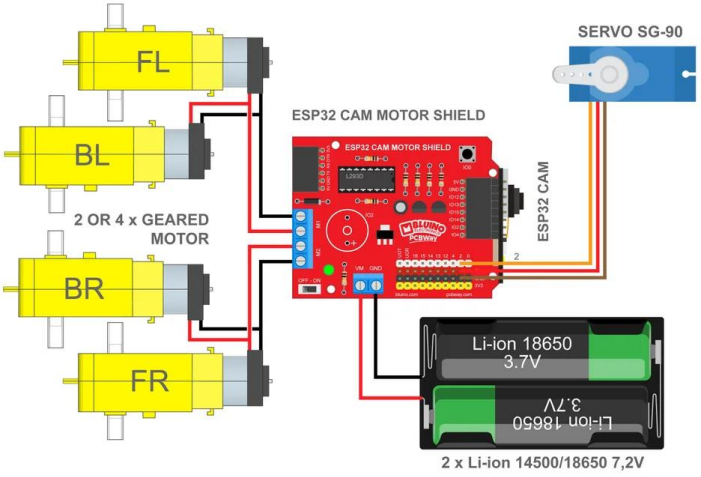
- In "Resources" of the last link download: Code for ESP32CAM Car, the code needed to make this car work, all in one ZIP file.
- To compile it use ESP32 Espressif System 1.0.6

- Follow instructions in about the hardware : https://dronebotworkshop.com/esp32cam-robot-car/
- Main electrical parts



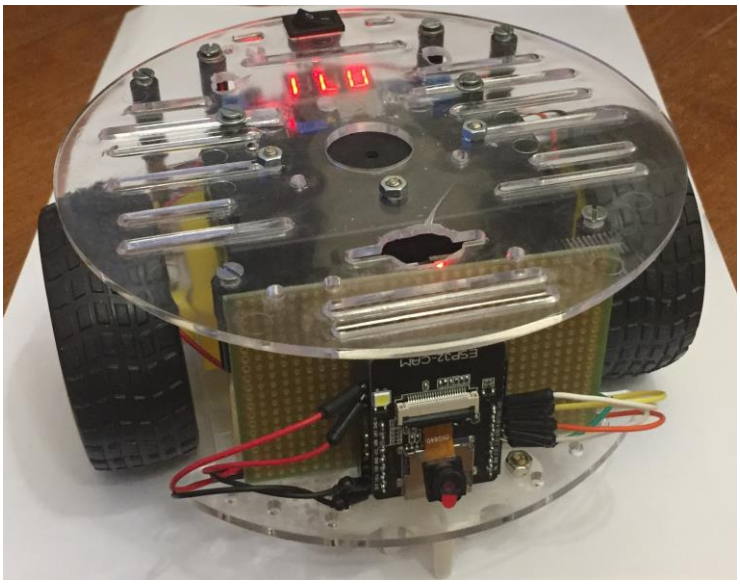
Camera and motor driver interconnection Motor and ESP32-CAM module power

- Power schematics



- Final results

Wifi Camera Robot Car – own implementation



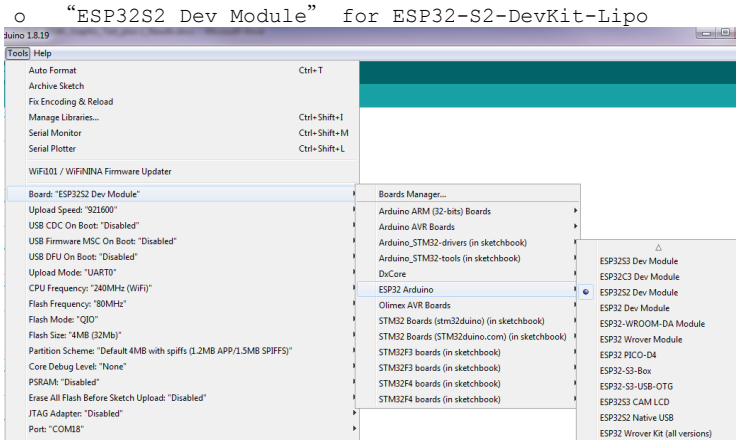
ESP32-S2 board on Arduino IDE

- For using ESP32-S2 boards like:

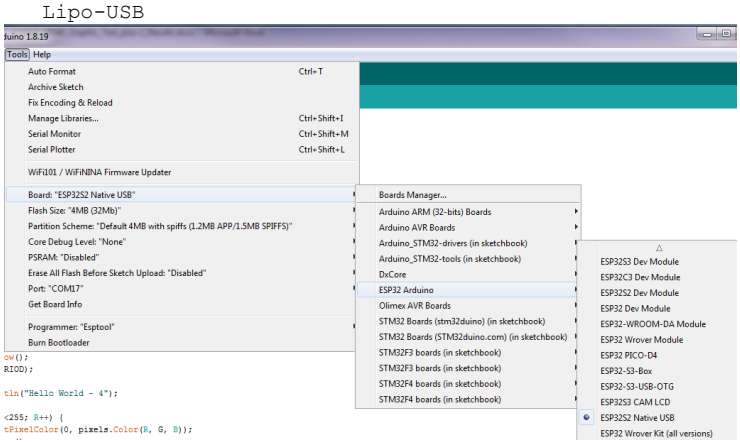
ESP32-S2-DevKit-Lipo ESP32-S2-WROVER-DevKit-Lipo-USB

- Install the ESP32-S2 support for Arduino IDE
- In "File" → "Preferences" add URL: https://espressif.github.io/arduino-esp32/package_esp32_index.json
- In "Tools" → "Boards" → "Board Manager" search for the esp32 platform and install ver. 2.0.0 or later

- Restart IDE and select board in "Tools" → "Board:
  - "ESP32S2 Dev Module" for ESP32-S2-DevKit-Lipo



  - "ESP32S2 Native USB" for ESP32-S2-WROVER-DevKit-Lipo-USB



- Connect ESP32-S2-DevKit-Lipo and install driver for USB-Serial CH340 adapter if needed
- Connect ESP32-S2-WROVER-DevKit-Lipo-USB and put it in boot loader's mode (hold GPIO0 low while reset)
- Install driver with Zadig software if needed
  - Enable in "Options" → "List all devices"
  - Choose device "ESP32-S2 (Interface 2)"
  - And option "USB Serial (CDC)"
- Any time for programming ESP32-S2-WROVER-DevKit-Lipo-USB has to be put in boot loader's mode and reset manually after uploading the sketch

Multitasking "Hello World & RGB LED" test

```
/*
 * Requires Adafruit NeoPixel library
 */
#include <Adafruit_NeoPixel.h>
#define PIN 18
#define NUMPIXELS 1
#define PERIOD 10 //ms
Adafruit_NeoPixel pixels(NUMPIXELS, PIN,
                         NEO_GRB + NEO_KHZ800);
int colors[3];
void setup() {
  pixels.begin();
  for (int i = 0; i < 3; i++) colors[i] = 0;
#if 1
  Serial.begin(115200); // ESP32-S2-DevKit-Lipo
#else
  Serial.begin();// ESP32-S2-WROVER-DevKit-Lipo-USB
  // Wait for serial port to connect.
  // Needed for native USB port only.
  while (!Serial) ;
#endif
  Serial.println("Hello World!");
  vTaskDelay(1000 / portTICK_PERIOD_MS);
  xTaskCreate(loop2,"loop2", 2048, NULL,1,NULL);
}
int n = 0;
void loop2( void * parameter ) {
  while(1) {
    Serial.print("Hello World - "); Serial.println(n++);
    vTaskDelay(2000 / portTICK_PERIOD_MS);
  }
}
void loop () {
  for (int i = 0; i < 3; i++) {
    int j;
    for (j = 0; j < 256; j++) {
      colors[i] = j;
      pixels.setPixelColor(0, pixels.Color(colors[0],
                           colors[1], colors[2]));
      pixels.show();delay (PERIOD);
    }
    for (j = 255; j >= 0; j--) {
      colors[i] = j;
      pixels.setPixelColor(0, pixels.Color(colors[0],
                           colors[1], colors[2]));
      pixels.show(); delay (PERIOD);
    }
  }
}
```

- Compiler messages for ESP32-S2-WROVER-DevKit-Lipo-USB

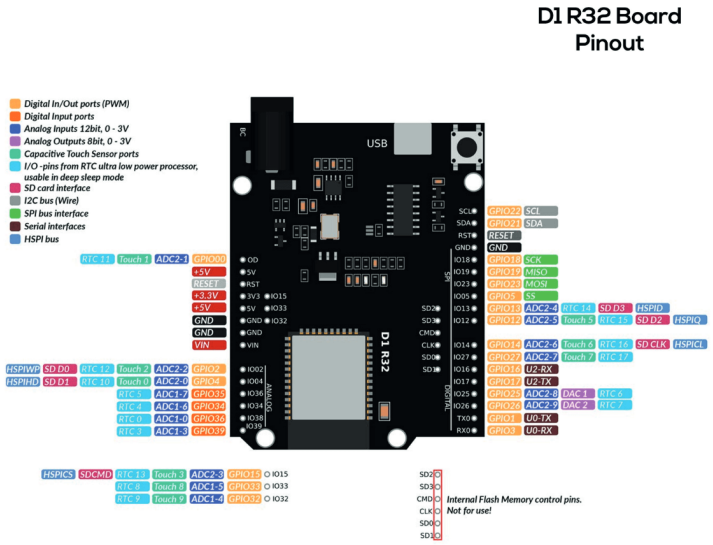
```
Sketch uses 291526 bytes (22%) of program storage space.
Maximum is 1310720 bytes.
Global variables use 27596 bytes (8%) of dynamic memory,
leaving 300084 bytes for local variables. Maximum is
327680 bytes.
```

- After running sketch on ESP32-S2-WROVER-DevKit-Lipo-USB composite device will be installed with TinyUSB DFU_RT, CDC and ESP32-S2 Firmware MSC devices.
- In terminal connected to USB-Serial CH340 following messages will be sent from ESP32-S2-DevKit-Lipo:

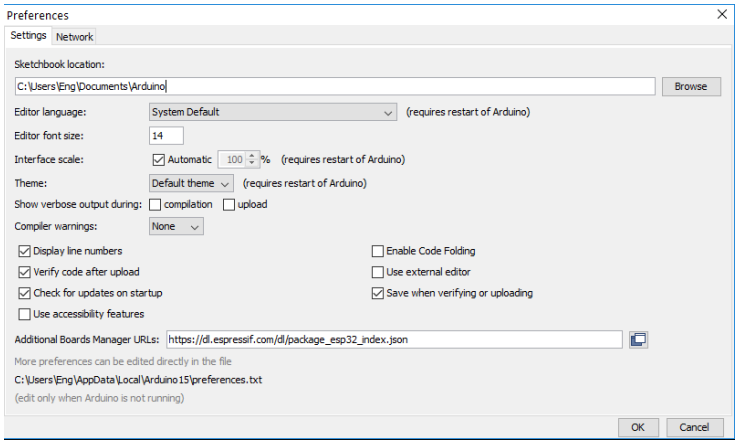
```
ESP-ROM:esp32s2-rc4-20191025                    System messages
Build:Oct 25 2019
rst:0x1 (POWERON),boot:0x8 (SPI_FAST_FLASH_BOOT)
SPIWP:0xee
mode:DIO, clock div:1
load:0x3ffe6100,len:0x524
load:0x4004c000,len:0xa70
load:0x40050000,len:0x2958
entry 0x4004c18c
Hello World!                                    Sent from setup section
Hello World - 0                                 Sent from loop2 task and will count every 2 sec
Hello World - 1
```

Arduino D1 R32 ESP32 board

- For using ESP32 boards like D1 R32

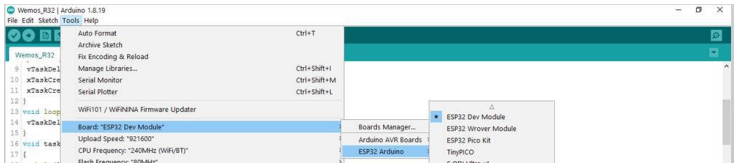


- In Preferences add URL: https://dl.espressif.com/dl/package_esp32_index.json

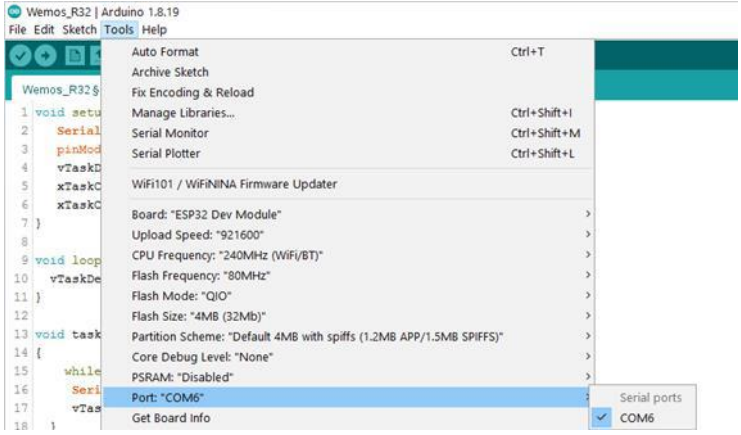


- Install esp32 in board manager

- Connect the board to Windows PC
- Install CH340 USB serial driver if needed and verify the port in "Device Manager": COM6 for example
- Install "ESP32 Dev Module" in board manager



- Setup USB serial port as verified above



Multitasking version of "Hello World & Blinking LED" test for ESP32

- Create new project "HelloWorld" and put the sketch:

```
void setup() {
  Serial.begin(112500);
  // By default the LED is connected to IO02
  pinMode(2, OUTPUT);
  // This will print default SPI pins
  Serial.println("Default SPI pins:");
  Serial.print("MOSI: "); Serial.println(MOSI);
  Serial.print("MISO: "); Serial.println(MISO);
  Serial.print("SCK: ");  Serial.println(SCK);
  Serial.print("SS: ");   Serial.println(SS);
  vTaskDelay(1000 / portTICK_PERIOD_MS);
  xTaskCreate(task1,"task1", 2048, NULL,1,NULL);
  xTaskCreate(task2,"task2", 2048, NULL,1,NULL);
}
void loop() {
  vTaskDelay(1000 / portTICK_PERIOD_MS);
}
void task1( void * parameter ) {
  while(1) {
    Serial.println("Hello World!");
    vTaskDelay(2000 / portTICK_PERIOD_MS);
  }
}
void task2( void * parameter) {
  while(1) {
    digitalWrite(2, HIGH);
    vTaskDelay(100 / portTICK_PERIOD_MS);
    digitalWrite(2, LOW);
    vTaskDelay(100 / portTICK_PERIOD_MS);
  }
}
```

- After compilation will see:

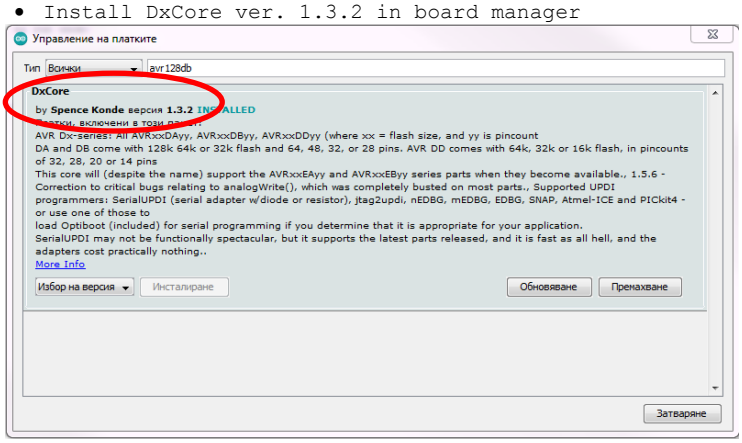
```
Sketch uses 204926 bytes (15%) of program storage space.
Maximum is 1310720 bytes.
Global variables use 13416 bytes (4%) of dynamic memory,
leaving 314264 bytes for local variables. Maximum is
327680 bytes.
```

- After uploading sketch will see fast blinking LED and following messages in terminal to USB serial port:

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x
00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1216
ho 0 tail 12 room 4
load:0x40078000,len:10944
load:0x40080400,len:6388
entry 0x400806b4
Default SPI pins:          Default settings belongs to VSPI
MOSI: 23
MISO: 19
SCK: 18
SS: 5
Hello World!               Will be repeated every 2 sec
Hello World!
```

AVR128db48 Arduino boards

- For using AVR128DB48 boards from Anton do:
- Add URL in Preferences:
  http://drazzy.com/package_drazzy.com_index.json
- Install DxCore ver. 1.3.2 in board manager



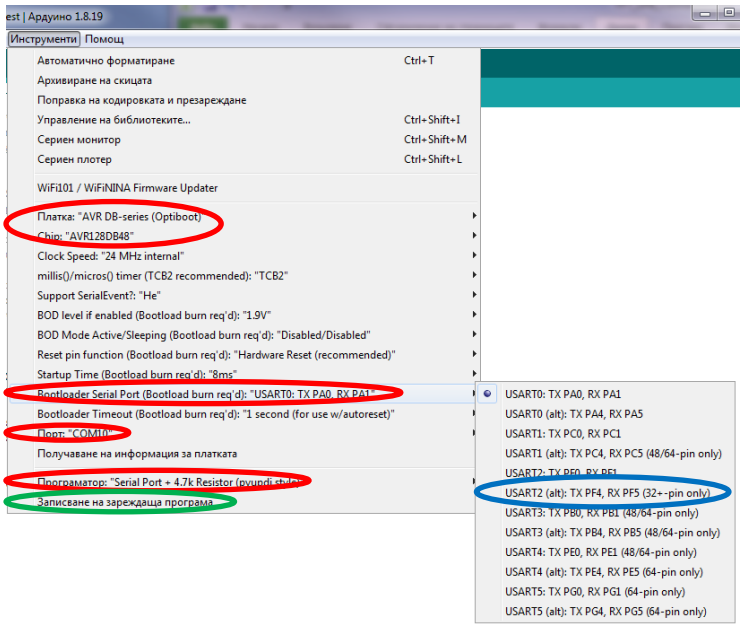
- Connect CP2102 USB to UART Bridge to Windows PC
- Install CP2102 USB driver if needed and verify the port: COM10 for example

UPDI programmer (to burn bootloader)

- Connect CP2102 USB to UART Bridge to the board
  - Rx ← 4.7k res. ← Tx ← AVR128DB48 UPDI (pin 41),
  - DTR → 200nF → RST (p. 40), GND, VCC (3.3V)
- Programmer: "Serial Port + 4.7k Resistor (pyupdi style)"
- Usage: Tools → Burn Bootloader
- Usage: Sketch → Upload Using Programmer

Regular serial programmer

- Connect CP2102 USB to UART Bridge to the board
  - CP2102/TTL-232R Tx → AVR128DB48 Rx0 (p. 45)
  - CP2102/TTL-232R Rx ← AVR128DB48 Tx0 (p. 44)
  - DTR → 200nF → RST (p. 40), GND, VCC (3.3V)
- Usage: Sketch → Upload

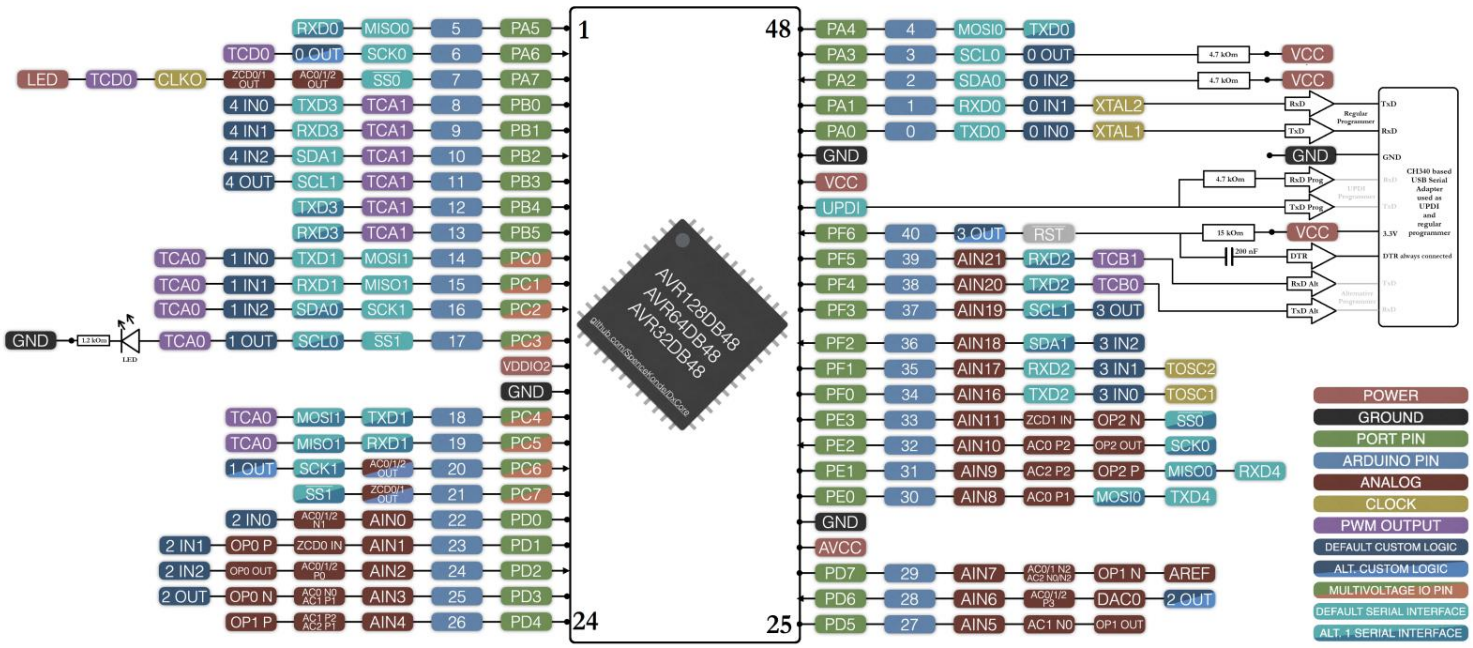


Bootloader serial port could be USART2 (alt), but using USART0 the PC COM port for both programming and serial communication with the sketch will be the same.

"Blinking LED" test for avr128db48

```
void setup() {
  // PIN_PC3 for avr128db48
  // may be different for other boards!
  pinMode(17, OUTPUT);
}
void loop() {
  digitalWrite(17, 1);
  delay(100);
  digitalWrite(17, 0);
  delay(100);
}
```

AVR128DB48 on 48 pin QFP adapter board



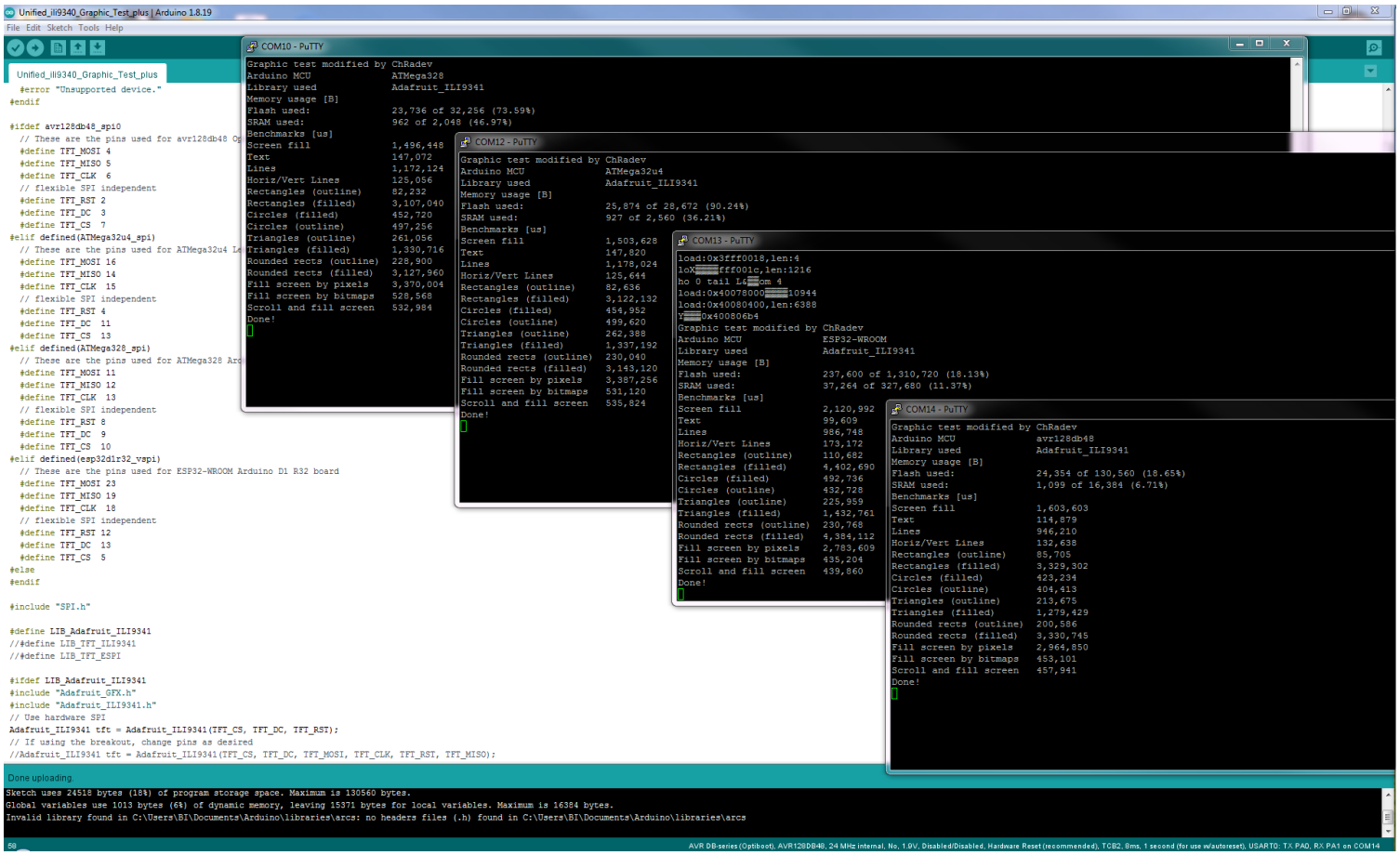
Other boards notes:

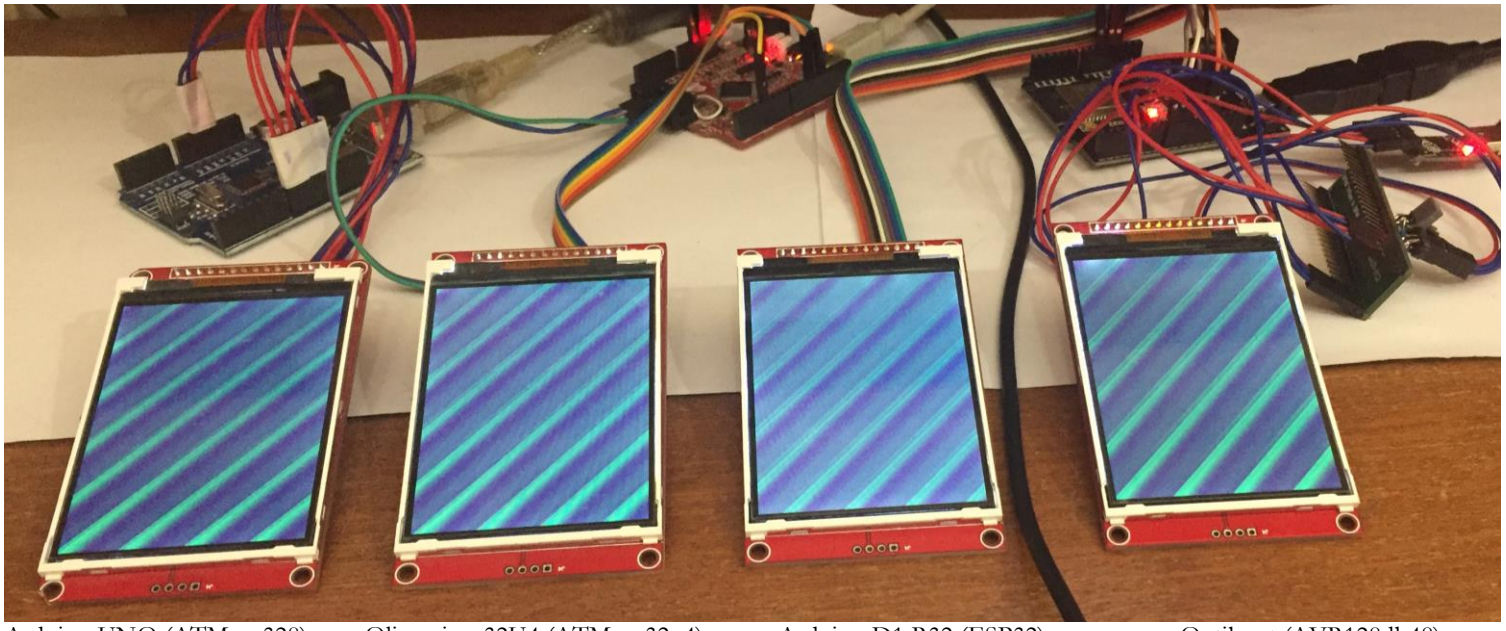
- Arduino UNO – install windows driver for USB-Serial CH340 adapter,
- Olimexino Nano – install windows driver for Arduino Leonardo compatible boards,
- Set in Tools → Board → Arduino AVR Boards → Arduino UNO or Arduino Leonardo respectively,
- Set in Tools → Port → corresponding COM port,
- LED pin may be different for different boards – change it in "Blinking LED" test sketch.

Connection setup for 3.2” 240x320 pixels TFT display with SPI interface

	3.2” TFT SPI LCD Display	Arduino UNO ATMega328	Olimexino32U4 ATMega32u4	Optiboot AVR128db48	Arduino R32 ESP-WROOM-32	Raspberry PI Pico RP2040 & CYW43439	Signal description (3.2” TFT SPI LCD Display)
1	VCC	VCC-3.3V	VCC-3.3V	VCC-3.3V	VCC-3.3V	VCC-3.3V (OUT)	3.3V power input (do not connect to 5V)
2	GND	GND	GND	GND	GND	GND	GND
3	CS	D10	D13	0,#SS, PA7	IO05	GP17	LCD chip select signal, low level enable
4	RESET	D8	D4	PA2 (0,SDA)	IO12	GP21	LCD reset signal, low level reset
5	DC/RS	D9	D11	PA3 (0,SCL)	IO13	GP20	LCD register / data selection signal, high level: register, low level: data
6	SDI(MOSI)	D11	D16	0,MOSI, PA4	IO23	GP16	SPI bus write data signal
7	SCK	D13	D15	0,SCK, PA6	IO18	GP18	SPI bus clock signal
8	LED	VCC-5V	VCC-5V	VCC-5V	VCC-5V	VCC-5V (VBUS)	Backlight control, high level lighting, if not controlled, connect 5V for always bright
9	SDO(MISO)	D12	D14	0,MISO, PA5	IO19	GP19	SPI bus read data signal, if you do not need to the read function, you cannot connect it

All 3 boards are connected to 3.2”SPI TFT display and running Unified graphic test





Arduino UNO (ATMega328) Olimexino-32U4 (ATMega32u4) Arduino D1 R32 (ESP32) Optiboot (AVR128db48)

Benchmark of unified graphic and scroll tests built on Adafruit_ILI9341, TFT_ILI9341 and TFT_eSPI libraries

Arduino board / MCU	UNO / ATMega328			Leonardo / ATMega32u4			D1 R32 / ESP32			Pi Pico / RP2040			Pi Pico / RP2040 (Overclocked)			AVR128db48
ILI9341 Library used	**Adafruit**	**TFT**	**Speed up**	**Adafruit**	**TFT**	**Speed up**	**Adafruit**	**TFT_eSPI**	**Speed up**	**Adafruit**	**TFT_eSPI**	**Speed up**	**Adafruit**	**TFT_eSPI**	**Speed up**	**Adafruit**
Memory usage [B]																
Flash used:	23,736 of 32,256 (73.59%)	21,870 of 32,256 (67.80%)		25,874 of 28,672 (90.24%)	23,992 of 28,672 (83.68%)		237,600 of 1,310,720 (18.13%)	269,072 of 1,310,720 (20.53%)		327,772 of 2,093,056 (15.65%)	372,092 of 2,093,056 (17.78%)		327,868 of 1,568,768 (20%)	372,180 of 1,568,768 (23%)		24,354 of 130,560 (18.65%)
SRAM used:	950 of 2,048 (46.39%)	746 of 2,048 (36.43%)		915 of 2,560 (35.74%)	711 of 2,560 (27.77%)		37,264 of 327,680 (11.37%)	36,864 of 327,680 (11.25%)		71,324 of 262,144 (27.21%)	71,768 of 262,144 (27.38%)		71,324 of 262,144 (27%)	71,768 of 262,144 (27%)		1,087 of 16,384 (6.63%)
Benchmarks [us]																
Screen fill	1,496,456	870,220	1.720	1,503,900	874,600	1.720	2,120,993	233,151	9.097	604,056	281,577	2.145	497,451	107,972	4.607	1,603,604
Text	147,088	60,416	2.435	147,820	60,724	2.434	99,610	15,346	6.491	45,452	18,831	2.414	30,599	8,085	3.785	114,885
Lines	1,172,116	242,732	4.829	1,178,004	243,988	4.828	986,748	89,909	10.975	454,856	101,897	4.464	304,234	42,741	7.118	946,199
Horiz/Vert Lines	125,064	71,336	1.753	125,656	71,696	1.753	173,171	20,128	8.603	50,042	23,541	2.126	40,853	9,078	4.500	132,637
Rectangles (outline)	82,228	45,844	1.794	82,632	46,076	1.793	110,682	12,727	8.697	32,657	14,932	2.187	26,417	5,773	4.576	85,703
Rectangles (filled)	3,107,060	1,807,436	1.719	3,122,844	1,816,740	1.719	4,402,687	484,050	9.096	1,253,856	584,372	2.146	1,032,576	224,086	4.608	3,329,307
Circles (filled)	452,728	284,064	1.594	454,916	285,536	1.593	492,735	63,960	7.704	167,914	71,149	2.360	126,969	28,025	4.531	423,221
Circles (outline)	497,252	135,580	3.668	499,604	136,148	3.670	432,728	33,343	12.978	199,626	37,258	5.358	133,263	15,561	8.564	404,412
Triangles (outline)	261,056	59,496	4.388	262,392	59,808	4.387	225,959	22,013	10.265	101,400	23,636	4.290	68,473	10,319	6.636	213,681
Triangles (filled)	1,330,720	694,456	1.916	1,337,200	698,032	1.916	1,432,757	164,864	8.691	429,998	195,995	2.194	345,244	75,450	4.576	1,279,412
Rounded rects (outline)	228,892	100,004	2.289	230,024	100,532	2.288	230,767	20,954	11.013	92,280	23,635	3.904	65,233	9,576	6.812	200,582
Rounded rects (filled)	3,127,968	1,976,936	1.582	3,143,588	1,987,180	1.582	4,384,111	487,395	8.995	1,257,871	586,292	2.145	1,032,024	225,027	4.586	3,330,751
Fill screen by pixels	3,369,992	918,732	3.668	3,387,308	923,492	3.668	2,783,609	835,657	3.331	1,255,234	504,753	2.487	805,373	229,258	3.513	2,964,859
Fill screen by bitmaps	528,576	855,088	0.618	531,112	859,520	0.618	435,203	840,458	0.518	66,438	520,180	0.128	70,363	234,904	0.300	453,099
Scroll and fill screen	**532,988**	**855,696**	**0.623**	**535,808**	**860,132**	**0.623**	**439,860**	**845,475**	**0.520**	**69,357**	**521,011**	**0.133**	**71,933**	**235,385**	**0.306**	**457,946**
Min	82,228	45,844		82,632	46,076		99,610	12,727		32,657	14,932		26,417	5,773		85,703
Avg	**1,097,346**	**598,536**	**1.833**	**1,102,854**	**601,614**	**1.833**	**1,250,108**	**277,962**	**4.497**	**405,402**	**233,937**	**1.733**	**310,067**	**97,416**	**3.183**	**1,062,687**
Max	3,369,992	1,976,936		3,387,308	1,987,180		4,402,687	845,475		1,257,871	586,292		1,032,576	235,385		3,330,751
Sum	16,460,184	8,978,036		16,542,808	9,024,204		18,751,620	4,169,430		6,081,037	3,509,059		4,651,005	1,461,240		15,940,298
DrawWithDMA test (bonsing of 42 colored and numbered circles)								**21.6 fps at CPU 240MHz SPI 27MHz**			**17.8 fps at CPU 133MHz SPI 27MHz**			**46.5 fps at CPU 250MHz SPI 62.5MHz**	**2.2 / 2.6**	

Notes:

- Memory usage numbers are as reported in runtime and slightly different than one reported by the compiler;
- Preparing of the data for filling the screen by pixels or bitmaps are made to be as fast as possible;
- Numbers for "Scroll and fill screen" tests at TFT_ILI9341 and TFT_eSPI libraries should be revised;
- At combination ESP32 and Adafruit_ILI9341 library SPI frequency was lowered to 3MHz;
- Numbers in "Speed up" column means the operation is that many times faster;
- Overclocking includes increasing of SPI and CPU speed up to 62.5MHz and 250MHz respectively and suggested solution by Bodemar in his Github issue 1460 (working reliably even with 30cm long wires).

Useful links for display drawing with DMA and speed assessment:

Raspberry Pi Pico with ILI9341 TFT and TFT_eSPI Arduino library using RAM & DMA
https://forum.arduino.cc/t/tft_espi-support-for-raspberry-pi-pico-added/702551
https://www.youtube.com/watch?v=njFXIzCTQ_Q
https://github.com/Bodmer/TFT_eSPI/issues/1460#issuecomment-1006661452

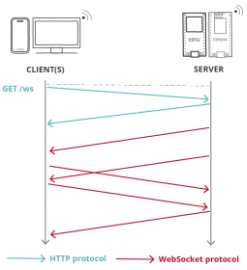
This application uses two sprites in RAM and DMA for filling display half buffer while updating the other half. The ILI9341 display operates reliably on Pi Pico up to 62.5MHz so frame rate up to ~43fps is possible with DMA. Overclocking CPU to 250MHz and applying Bodmer note makes it possible frame rates to go up to 46.5fps. The total consumption in overclocked mode of both Pi Pico and SPI TFT is 110mA.

Network performance using AsyncWebServer and AsyncTCP libraries on Pi Pico W and ESP32 series of boards

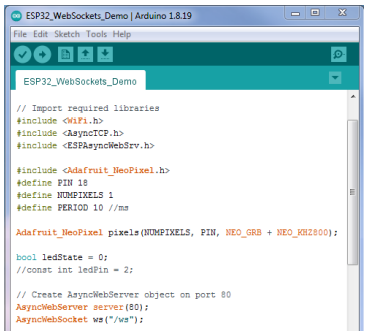
Startup projects working on ESP32 S2 Olimex boards and based on ESPAsyncWebServer library for Arduino:

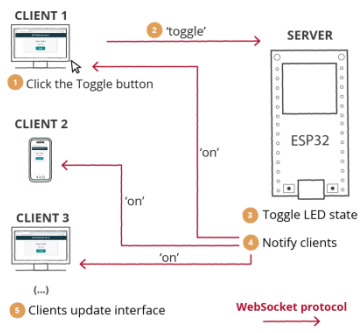
- ESP32: Create a Wi-Fi Manager (AsyncWebServer library)

Application uses SPIFS on ESP32 systems to hold web and configuration files which have to be written manually by "ESP32 Sketch Data Upload" tool of Arduino IDE. The application first runs in AP mode asking for connection credentials of the local router. After storing them in FS files and restart it runs in STA mode. Main web page allows controlling built-in LED.



- ESP32 WebSocket Server: Control Outputs (Arduino IDE)

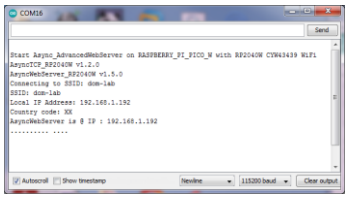


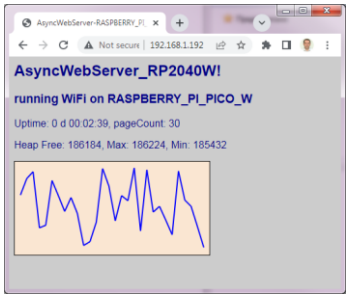


Application on ESP32 runs in STA mode with credentials defined in the sketch and open WebSocket server to control the LED. Its status can be changed by any client and will be updated at all the clients.

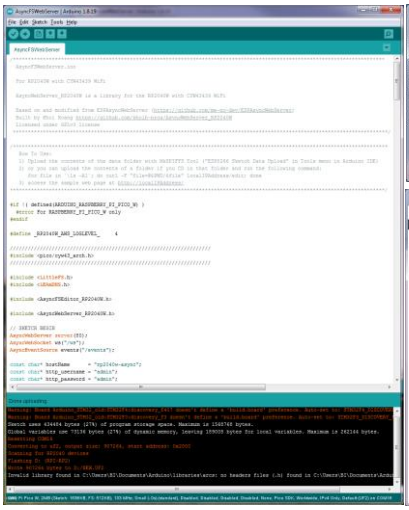
It was used Adafruit NeoPixel library to run above projects on Olimex ESP32 S2 series of boards with RGB instead of regular LED.

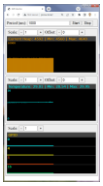
Startup projects on Raspberry Pi Pico W – AsyncWebServer for RP2040W library examples:

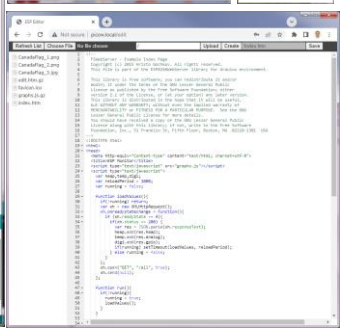




- Async_AdvancedWebServer



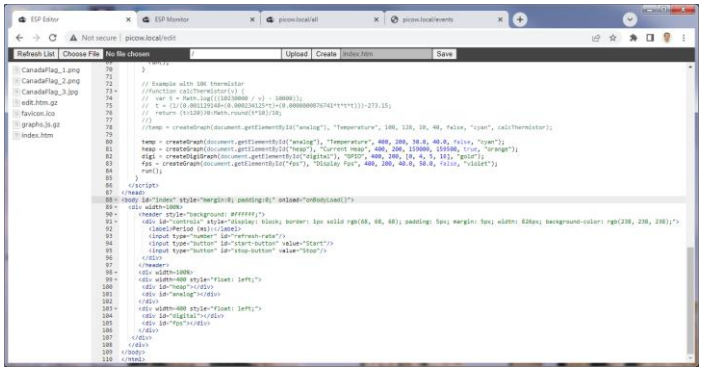




- AsyncFSWebServer (library ver. 1.5.0 did not run out of the box)

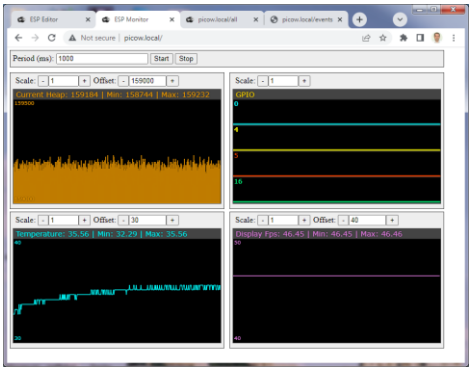
Application uses LittleFS library to access SPI flash FS. It also uses mDNS, basic authentication, AsyncWebSocket, AsyncEventSource and AsyncFSEditor_RP2040W library to show and edit files.

AsyncFSWebServer and DrawWithDMA combined multicore application for Raspberry Pi Pico W was done by simply putting both files in a single project, renaming setup and loop functions in the second file to setup1 and loop1 and commenting the line Serial.begin(115200). Display drawing (42 circles) speed was the same (17.85fps) without appreciable change in the web access. Temperature measured by internal sensor is increased with approximately 2°C (up to 31°C). The heap is increased from 5kB up to 159kB. CPU overclocking to 250MHz did not speed up display drawing and web access but increase the temperature with approximately 3°C (up to 34°C). This makes CPU overclocking useless. SPI speed can be changed in User_Setup.h of TFT_eSPI library. Changing it from 27MHz to 55MHz (2x) did not speed up display drawing but thanks to Bodmer comment and CPU clocking at 125MHz (SPI clock is 62.5MHz) display drawing can be speed up to 43-45fps @42 circles and 46.3fps @36 circles. Overclocking CPU to 250MHz (probably SPI clock is again 62.5MHz) increase display drawing speed up to 46.5fps @42 circles (2.6x) while working smoothly and reliably. Total consumption is increased form 110mA in case of overclocked DrawWithDMA single core application up to 144mA for combined multicore application.



Remote file manager and editor

+



On-line monitor

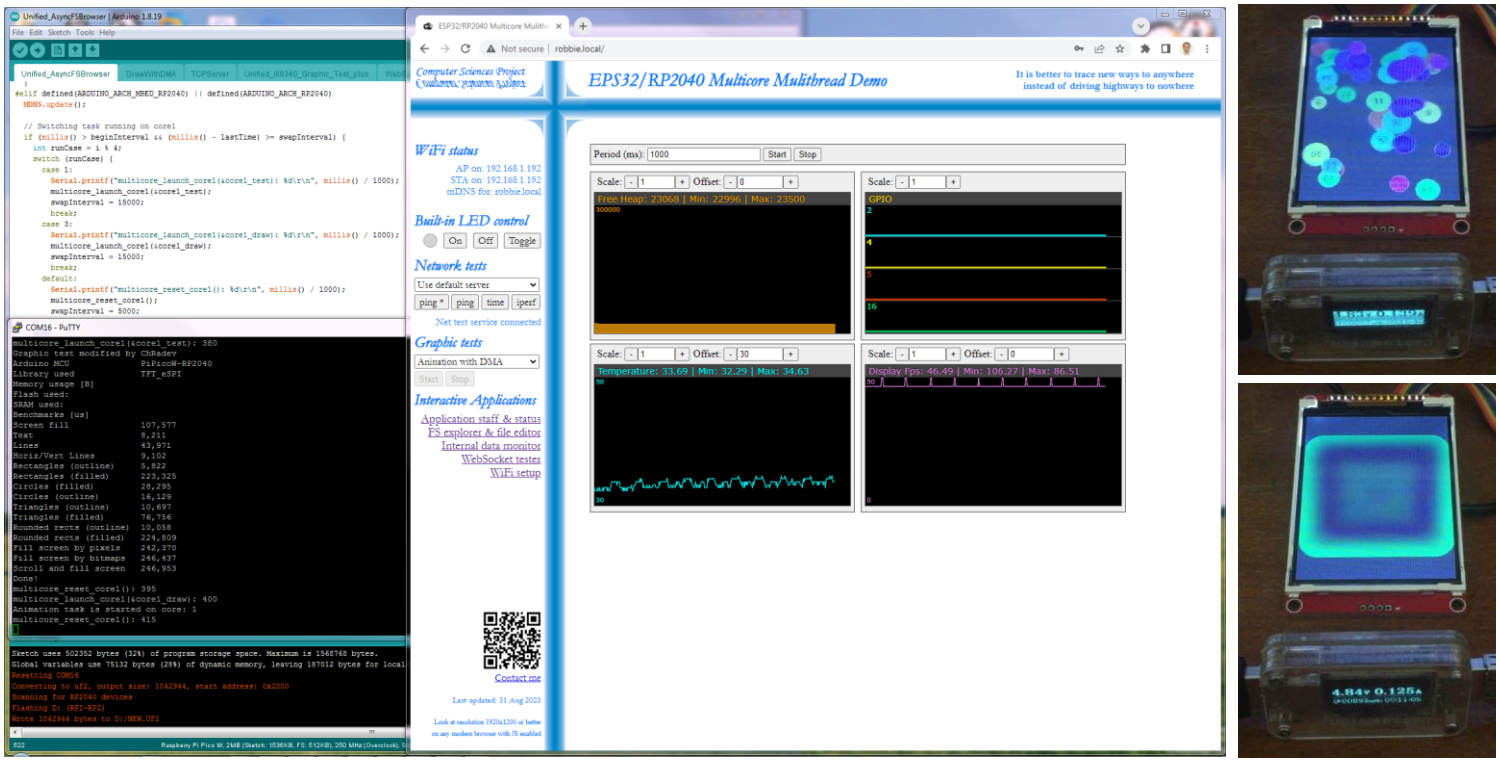
+

SPI TFT display

AsyncWebServer for RP2040W library built by Khoi Hoang is based on and modified from ESPAsyncWebServer library support of ESP32 and ESP8266 on Arduino cores. Next steps to be done:

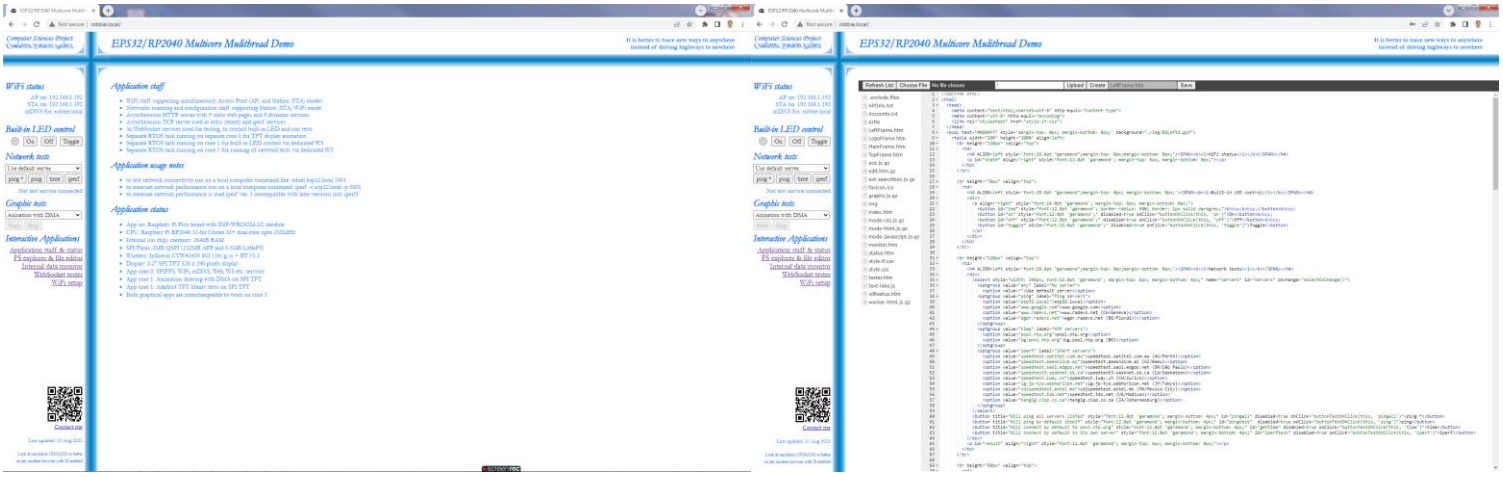
- Check code compatibility for both ESP32 and Pi Pico W boards;
- Build unified web server application working in AP and STA modes and including WiFi management, mDNS, LittleFS and WebSockets;
- Dynamically running of different tasks on the second CPU core.

Unified application for ESP32 and RP2040 based on AsyncFSWebBrowser and many other things

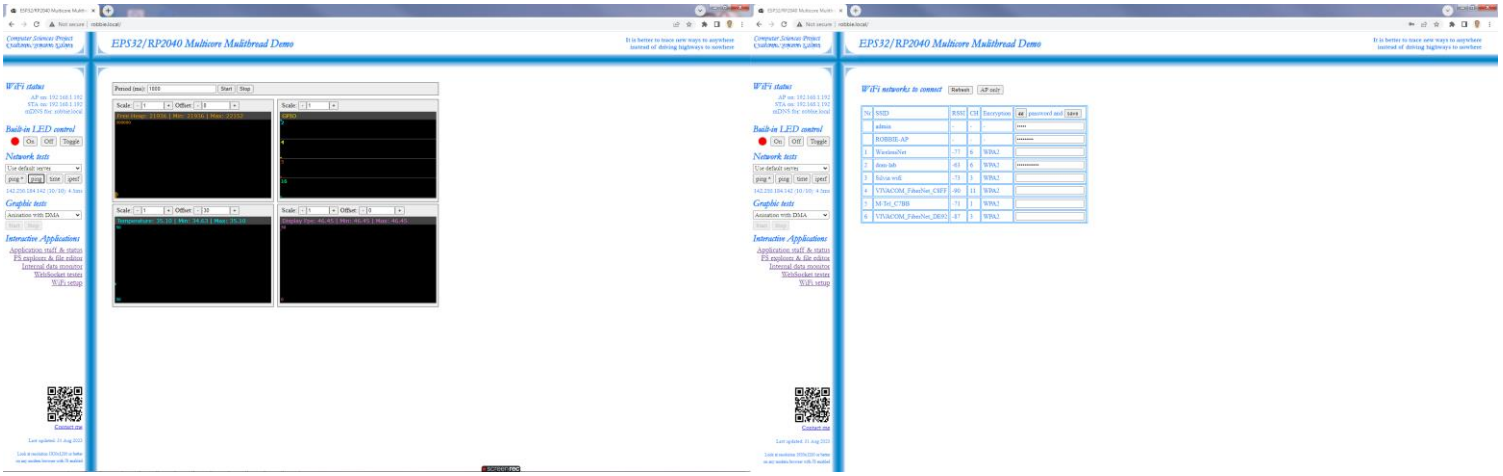


Arduino IDE with multi-file Unified_AsyncFSBrowser project for ESP32 and RP2040 based boards, console with printout of the Adafruit TFT tests adapted for eTFT library and log of swapping graphical tasks to work on core1 and web application with internal monitor showing graphs of the free heap memory, the GPIO states, the internal temperature sensor and the animation frame rate. On the right are shown pictures of graphical tasks (animation and graphical tests) running alternatively on core 1.

The most attractive application feature is complete independency of the performance of tasks running on different CPU cores. Other impressive result is graphical performance of animation task (46 frames per second at 42 bouncing circles) and all Adafruit TFT tests adapted to work with the eTFT library (especially scrolling display speed of 0.8ms per 240 pixels line).



Home view of the web application with control staff and system information (on the left) and control staff with internal FS file viewer and editor (on the right)



Other views with graphs of the internal system and the application parameters (on the left) and WiFi setup page (on the right). It is evident that the results of the LED toggling and the network ping command are synchronized on both web clients while the action is taken from different one.