TARJETA DE DESARROLLO (RP2040 & W5500) BASADA EN RASPBERRY PI PICO CONECTIVIDAD ETHERNET

W55RP20-EVB-PICO

Productos evaluados por **ingenieros calificados**

Garantía y seguridad en cada producto

Experiencia de compra en la **calidad** como sello distintivo

Descripción:

W55RP20-EVB-Pico es una placa de evaluación para W55RP20, un chip que combina W5500, un controlador TCP/IP cableado, y RP2040. Por lo tanto, están disponibles tanto las funciones de Raspberry Pi Pico como las del W5500.

- Clon de Raspberry Pi Pico
- Ethernet (CHIP TCP/IP cableado W55RP20)

Especificaciones:

Microcontrolador W55RP20

- Memoria Flash interna de 2MB.
- Procesador Cortex M0+ de doble núcleo a hasta 133MHz.
- Memoria SRAM multibanco de alto rendimiento de 264kByte.
- Memoria Flash Quad-SPI externa con ejecución en el lugar (XIP).
- Tejido de barra transversal completa de alto rendimiento para bus.
- 22 E/S de propósito general multifunción (se pueden utilizar 4 para ADC).
  - Voltaje de E/S de 1.8 a 3.3 V (NOTA: el voltaje de E/S de Pico está fijado en 3.3 V).
- Convertidor analógico a digital (ADC) de 12bits y 500kbps.
- Varios periféricos digitales.
  - 2 UART, 2 I2C, 2 SPI, 16 canales PWM.
  - 1 Temporizador con 4 alarmas, 1 Contador de tiempo real.

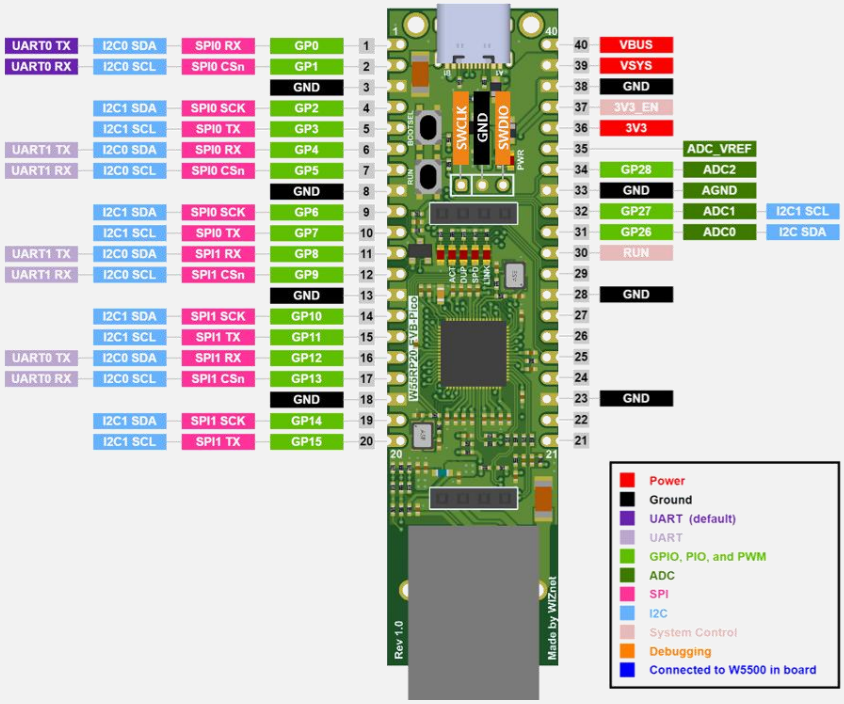
- 2 bloques de E/S programables (PIO), 8 máquinas de estado.
- E/S de alta velocidad, flexible y programable por el usuario.
- Puede emular interfaces como tarjeta SD y VGA.
- Admite protocolos de Internet cableados: TCP, UDP, ICMP, IPv4, ARP, IGMP, PPPoE.
- Admite 8 sockets de hardware independientes simultáneamente.
- Memoria interna de 32 Kbytes para buffers TX/RX.
- Admite interfaz periférica serial de alta velocidad (SPI MODE 0, 3).

Puerto USB C para alimentación y datos (y para reprogramar memoria Flash)
Puerto de depuración de cable serial ARM (SWD) de 3 pines
10/100 Ethernet PHY integrado
Admite negociación automática

- Half/Full duplex
- 10/100 basado

RJ45 incorporado (POE)
DC-DC incorporado (PWM/PFM)
Módulos adicionales instalados para habilitar POE

Hardware:



La distribución de pines del W55RP20-EVB-Pico está conectada directamente al GPIO de RP2040, como se muestra en la imagen anterior. Tiene la misma distribución de pines que la placa Raspberry Pi Pico. Sin embargo, GPIO17, GPIO20, GPIO21, GPIO22, GPIO23, GPIO24 y GPIO25 no están disponibles debido a la conexión dentro de la placa.

Los pines GPIO W55RP20 utilizados dentro de W55RP20-EVB-Pico son los siguientes.

E/S	Pin	Descripción
O	GPIO16	Pin de selección de modo DC-DC
I	GPIO18	Detección VBUS: alta si VBUS está presente, baja en caso contrario
O	GPIO19	Conectado al LED del usuario
I	GPIO29	Se utiliza en modo ADC (ADC3) para medir VSYS/3

Además de los pines GPIO y de tierra, hay otros 7 pines en la interfaz principal de 40 pines:

N.º Pin	Nombre del pin	Descripción
PIN40	VBUS	Voltaje de entrada micro-USB, conectado al pin 1 del puerto micro-USB. Nominalmente 5V.
PIN39	Sistemas de visión	Voltaje de entrada del sistema principal, que puede variar dentro del rango permitido de 4.3V a 5.5V, y es utilizado por el LDO integrado para generar los 3.3V.
PIN37	3V3_ES	Se conecta al pin de habilitación LDO integrado. Para desactivar los 3.3V (que también desenergizan el RP2040 y el W5500), ponga este pin en cortocircuito.
PIN36	3 contra 3	Alimentación principal de 3.3V a RP2040 y W5500, generada por el LDO integrado.
PIN35	ADC_VREF	El voltaje de fuente de alimentación (y referencia) del ADC se genera en el W5500-EVB-Pico filtrando el suministro de 3.3V.
PIN33	AGND	Referencia de tierra para GPIO26-29.

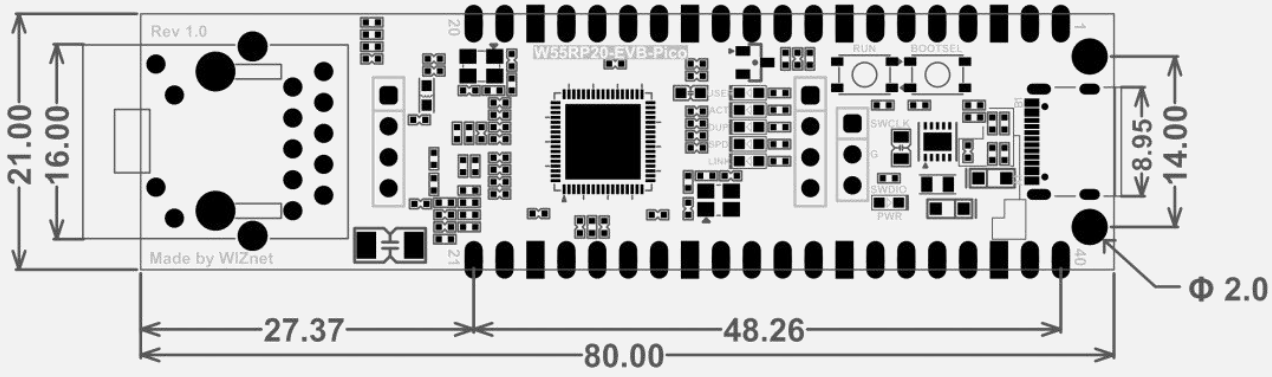
N.º Pin	Nombre del pin	Descripción
PIN30	CORRER	Pin de habilitación RP2040, para restablecer RP2040, ponga este pin en cortocircuito.

Condiciones de operación:

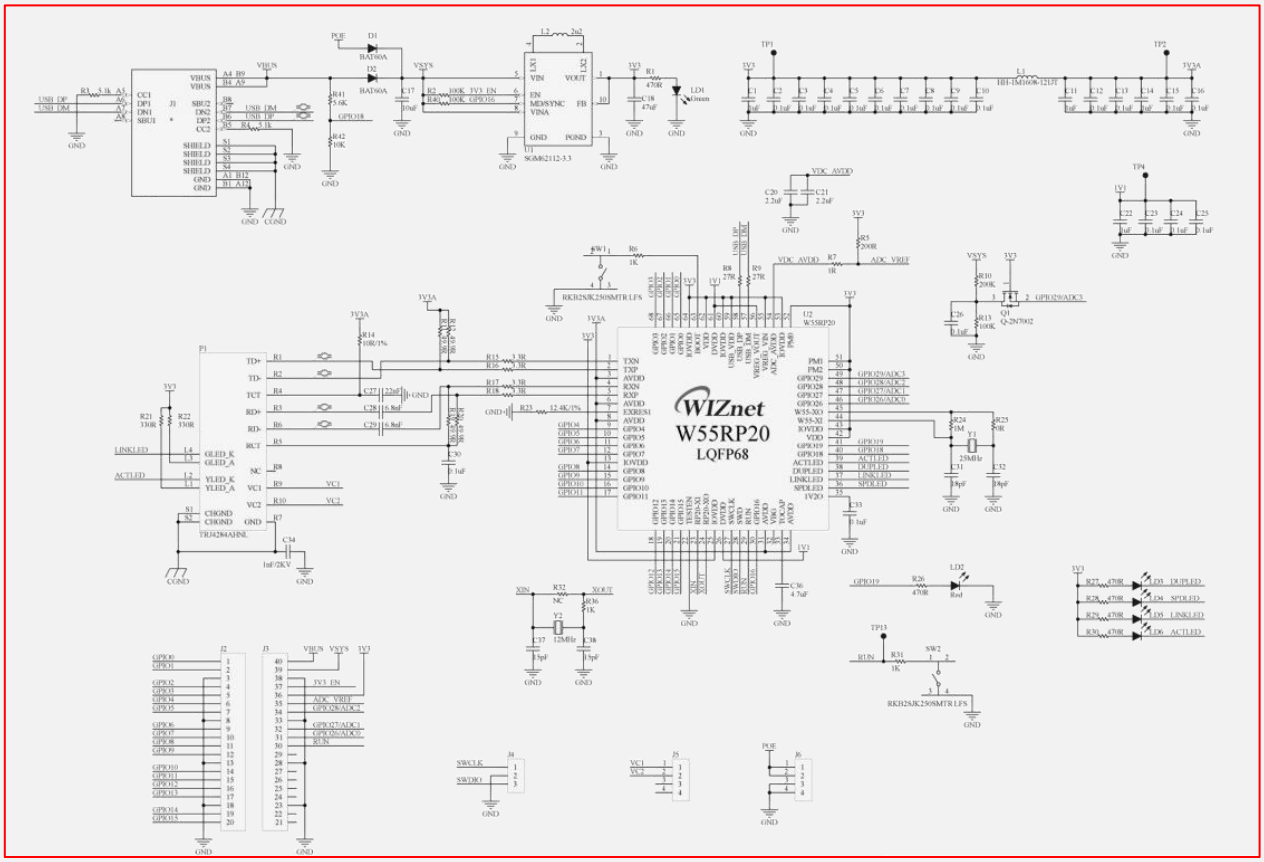
Artículo	Descripción
Temperatura de funcionamiento MÁXIMA	85 °C
Temperatura de funcionamiento MÍNIMA	-45°C
VBUS	5VCC (+/- 10 %)
VSYS Mínimo	4.3VCC
VSYS Máximo	5.5VCC

La temperatura ambiente máxima de funcionamiento recomendada es de 70 °C.

Dimensiones:



Esquema:



Código de ejemplo:

```
import board
import rp2pio
import adafruit_pioasm
import digitalio
import time

from wiznet5k_pio import WIZNET5K
from adafruit_ticks import ticks_ms, ticks_diff
from micropython import const

from wiznet5k_pio import (
    SNSR_SOCK_ESTABLISHED,
```

```
    SNSR_SOCK_CLOSE_WAIT,
    SNSR_SOCK_LISTEN,
    SNSR_SOCK_CLOSED,
)

# PIO assembly code: SPI master implementation
spi_master = """
.program spi_master
    pull block
    set x, 7
bitloop:
    out pins, 1
    set pins, 1
    nop [1]
    in pins, 1
    set pins, 0
    jmp x-- bitloop
    push block
"""

mac_address = [0x00, 0x08, 0xDC, 0x01, 0x02, 0x03]
ip_address = [192, 168, 11, 110]
gateway_ip = [192, 168, 11, 1]
subnet_mask = [255, 255, 255, 0]

# PIO and State Machine setup
assembled = adafruit_pioasm.assemble(spi_master)
sm = rp2pio.StateMachine(
    assembled,
    frequency=1_000_000,
```

```
    first_out_pin=board.GP23,  # mosi
    first_in_pin=board.GP22,  # miso
    first_set_pin=board.GP21,  # clk
    out_pin_count=1,
    in_pin_count=1,
    set_pin_count=1,
    in_shift_right=False,
    out_shift_right=False,
    push_threshold=8,
    pull_threshold=8,
)

# CS and RST pin setup
cs_pin = digitalio.DigitalInOut(board.GP20)
rst_pin = digitalio.DigitalInOut(board.GP25)

# Initialize WIZNET5K
wiznet = WIZNET5K(
    sm,
    cs_pin,
    rst_pin,
    mac_address=mac_address,
    ip_address=ip_address,
    gateway_ip=gateway_ip,
    subnet_mask=subnet_mask,
)
# After creating the WIZNET5K instance
version = wiznet.read_version()
print(f"W5500 Version: 0x{version:02X}")
print("Network information :", wiznet.ifconfig)
```

```
## TCP
# Initialize socket and start listening (MAX socket < 8)
sock_num_tcp = 0
port_tcp = 5000

print("Socket Open (TCP)")
wiznet.socket_init(sock_num_tcp)
wiznet.socket_listen(sock_num_tcp, port_tcp)

print(f"Listening on TCP port {port_tcp}")

while True:
    # Handle TCP socket
    status_tcp = wiznet.socket_status(sock_num_tcp)
    if status_tcp == SNSR_SOCK_ESTABLISHED:  # SOCK_ESTABLISHED
        # Receive data from the client
        data = wiznet.socket_recv(sock_num_tcp)
        if data:
            remote_port = wiznet.remote_port(sock_num_tcp)
            print(f"[TCP] Dest P: {remote_port} Received: {data}")
            # Echo the data back to the client
            wiznet.socket_send(sock_num_tcp, data)
    elif status_tcp == SNSR_SOCK_CLOSE_WAIT:  # SOCK_CLOSE_WAIT
        print("[TCP] Client disconnected, closing socket")
        wiznet.socket_close(sock_num_tcp)
        wiznet.socket_init(sock_num_tcp)
        wiznet.socket_listen(sock_num_tcp, port_tcp)
    elif status_tcp == SNSR_SOCK_LISTEN:  # SOCK_LISTEN
        pass  # Listening for incoming connections
```

```
elif status_tcp == SNSR_SOCK_CLOSED:  # SOCK_CLOSED
    wiznet.socket_init(sock_num_tcp)
    wiznet.socket_listen(sock_num_tcp, port_tcp)
```

Enlaces externos:

Earlephilhower. (s. f.). *GitHub - earlephilhower/arduino-pico: Raspberry Pi Pico Arduino core, for all RP2040 and RP2350 boards*. GitHub. https://github.com/earlephilhower/arduino-pico

WIZnet-ioNIC. (s. f.). *GitHub - WIZnet-ioNIC/WIZnet-ioNIC-Circuitpython*. GitHub. https://github.com/WIZnet-ioNIC/WIZnet-ioNIC-Circuitpython

WIZnet-ioNIC. (s. f.-b). *GitHub - WIZnet-ioNIC/WIZnet-ioNIC-micropython: MicroPython - a lean and efficient Python implementation for microcontrollers and constrained systems*. GitHub. https://github.com/WIZnet-ioNIC/WIZnet-ioNIC-micropython

WIZnet-ioNIC. (s. f.-c). *GitHub - WIZnet-ioNIC/WIZnet-PICO-AWS-C: AWS IoT SDK Example for RP2040, RP2350*. GitHub. https://github.com/WIZnet-ioNIC/WIZnet-PICO-AWS-C

WIZnet-ioNIC. (s. f.-d). *GitHub - WIZnet-ioNIC/WIZnet-PICO-C: Ethernet Example for RP2040, RP2350*. GitHub. https://github.com/WIZnet-ioNIC/WIZnet-PICO-C