INCLUSIÓN DEL MICROCONTROLADOR RASPBERRY PI PICO RP2040 PARA LA TRANSCOMPILACIÓN DE SU CÓDIGO FUENTE DESDE EL LENGUAJE BASADO EN V A SU COMPILADOR NATIVO C.

AUTOR:
ANDERSON CAMILO GUTIÉRREZ ÁLVAREZ

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

UNIVERSIDAD DISTRITAL "FRANCISCO JOSÉ DE CALDAS"
FACULTAD TECNOLÓGICA – INGENIERÍA ELÉCTRICA POR CICLOS PROPEDÉUTICOS

BOGOTÁ

2024

INCLUSIÓN DEL MICROCONTROLADOR RASPBERRY PI PICO RP2040 PARA LA TRANSCOMPILACIÓN DE SU CÓDIGO FUENTE DESDE EL LENGUAJE BASADO EN V A SU COMPILADOR NATIVO C.

AUTOR:
ANDERSON CAMILO GUTIÉRREZ ÁLVAREZ

TRABAJO DE GRADO

MODALIDAD: PASANTÍA

FERNANDO MARTÍNEZ SANTA

UNIVERSIDAD DISTRITAL “FRANCISCO JOSÉ DE CALDAS”
FACULTAD TECNOLÓGICA – INGENIERÍA ELÉCTRICA POR CICLOS PROPEDÉUTICOS

BOGOTÁ

2024

DEDICATORIA

A mi familia, compañeros y profesores,

Dedico este trabajo mi familia por ser el pilar más importante y demostrarme siempre su cariño y apoyo incondicional sin importar muestras diferencias de opiniones. A mi hermano menor que me inspira a hacer mejor las cosas y ser un buen ejemplo para él. A mis compañeros por ser un gran equipo y cumplir con este reto tan importante que nos ha formado como profesionales y como personas. Y por último quiero dedicar este trabajo a los profesores que dedican su vida a la enseñanza y lo hacen con esmero y dedicación.

Contenido

CONTENIDO ..... *¡Error! Marcador no definido.*
LISTA DE ESQUEMAS ***4***
LISTA DE TABLAS ***4***
LISTA DE FIGURAS ***4***
INTRODUCCIÓN. ***6***
1. OBJETIVOS ***7***
2. PLAN DE TRABAJO ***7***
3. MARCO TEÓRICO ***9***
4. DESARROLLO DE PASANTÍA ***11***
5. CONCLUSIONES Y RECOMENDACIONES ***46***
6. BIBLIOGRAFÍA ..... *¡Error! Marcador no definido.*

LISTA DE ESQUEMAS

ESQUEMA 1 ..... *10*
ESQUEMA 2 ..... *10*
Esquema 3: Diagrama interactivo de Aixt *21*
Esquema 4: Estructura de carpetas y archivos *22*
Esquema 5: Estructura de subcarpetas y archivos en api *23*
Esquema 6: Estructura en la seccion SETUP *30*
Esquema 7: Estructura en sección PROJECTS *32*

LISTA DE TABLAS

Tabla 1: Plan de trabajo *7*
Tabla 2: Cronograma *8*
Tabla 3: Desarrollo de la pasantía *12*

LISTA DE FIGURAS

Figura 1: Código de programación parpadeo led integrado *13*
Figura 2 Código programación: Entrada y salida digital *14*
Figura 3: Código programación convertidor análogo a digital *15*
Figura 4: Código programación salida PWM *16*
Figura 5: Código programación comunicación serial *17*
Figura 6: Código en archivo - pin.c.v – modulo pin *23*
Figura 7: Código en archivo - setup.c.v – modulo pin *24*
Figura 8: Código en archivo - high.c.v – modulo pin *24*
Figura 9: Código en archivo - low.c.v – modulo pin *24*
Figura 10: Código en archivo - read.c.v – modulo pin *25*
Figura 11: Código en archivo - write.c.v – modulo pin *25*

Figura 12: Código en archivo - adc.c.v – módulo adc 25
Figura 13: Código en archivo - read.c.v – módulo adc 26
Figura 14: Código en archivo - pwm.c.v – módulo pwm 26
Figura 15: Código en archivo - write.c.v – módulo pwm 26
Figura 16: Código en archivo - uart.c.v - módulo uart 27
Figura 17: Código en archivo - any.c.v - módulo uart 27
Figura 18: Código en archivo - setup.c.v - módulo uart 27
Figura 19: Código en archivo - read.c.v - módulo uart 28
Figura 20: Código en archivo - print.c.v - módulo uart 28
Figura 21: Código en archivo - println.c.v - módulo uart 28
Figura 22: Código en archivo - builtin.c.v 29
Figura 23: Código en archivo - Raspberry-Pi-Pico.toml 32
Figura 24: Código en archivo - settings.json 33
Figura 25: Código en archivo - DIGITAL.v - sección PROJECTS 33
Figura 26: Código en archivo - DIGITAL.ino - sección PROJECTS 35
Figura 27: Código en archivo - ADC.v - sección PROJECTS 36
Figura 28: Código en archivo - ADC.ino - sección PROJECTS 39
Figura 29: Código en archivo - PWM.v - sección PROJECTS 40
Figura 30: Código en archivo - PWM.ino - sección PROJECTS 43
Figura 31: Código en archivo - UART.v - sección PROJECTS 43
Figura 32: Código en archivo - UART.ino - sección PROJECTS 46

INTRODUCCIÓN.

El grupo de investigación ARMOS apoya el área de electrónica de potencia de los programas Tecnología en Sistemas de Potencia de Media y Baja Tensión e Ingeniería Eléctrica por ciclos, Facultad Tecnológica, de la Universidad Distrital Francisco José de Caldas. Actualmente el grupo desarrolla proyectos de investigación y desarrollo en las áreas de electrónica, instrumentación, control, automatización, robótica, procesamiento de señales, gestión de energía, fuentes renovables y capacitación en investigación e ingeniería. (Fernando Martinez Santa - Fredy Martinez - GRUPO ARMOS, s.f.)

Este proyecto está dirigido a la programación de microcontroladores, la cual tiene algunos aspectos que pueden dificultar su trabajo, uno puede ser el punto de vista económico o de recursos, y el otro aspecto seria su lenguaje de programación. Con el proyecto Aixt se busca utilizar una sintaxis de lenguaje moderno, enfocándose en dispositivos de bajos recursos.

El proyecto consistirá en darle soporte a la serie de microcontroladores RP2040 (Raspberry Pi Pico) para integrarse al proyecto Aixt, esto se realizará implementando una API en C incluyendo las funciones básicas de funcionamiento del microcontrolador soportadas por el proyecto Aixt y un archivo de configuración que permita al transcompilador hacer una traducción efectiva entre los dos lenguajes. El proyecto permitirá realizar código fuente en lenguaje Aixt que sea correctamente compilado a la serie RP2040 de manera automática, sin tener que realizar cambios manuales que puedan resultar tediosos y costosos en términos de tiempo y recursos. Aixt está optimizado para funcionar eficientemente en dispositivos con recursos limitados como los son los microcontroladores de la serie RP2040, lo que lo hace adecuado para aplicaciones que requieren una solución eficiente en términos de recursos.

Raspberry Pi Pico es una tarjeta de desarrollo basada en el microcontrolador RP2040 diseñado y construido por los ingenieros de Raspberry Pi. Esta tarjeta ha sido diseñada para ser una plataforma de desarrollo muy flexible y de bajo costo, que puede tener aplicaciones en Domótica, robótica y sistemas de control, entre otros.

1. OBJETIVOS

1.1 OBJETIVO GENERAL

Dar soporte al proyecto Aixt para la integración de los microcontroladores de la serie RP2040.

1.2 OBJETIVOS ESPECÍFICOS

- Dar soporte a un lenguaje moderno basado en V para ser transcompilado de manera efectiva en lenguaje C para microcontroladores de la serie RP2040.
- Desarrollar un conjunto de funciones básicas en C para microcontroladores de la serie RP2040, que pueda ser usado desde un lenguaje basado en V luego de su transcompilado.
- Realizar una documentación que permita a los desarrolladores y personas interesadas entender la estructura, funcionalidades y características del código, lo que a su vez también les permitirá contribuir al proyecto si así lo desean, puesto que se desea que sea de código abierto

2. PLAN DE TRABAJO

PLAN DE TRABAJO

Actividad	**Descripción actividad**
API	En esta etapa se debe realizar una recopilación de funciones básicas (entrada y salida de datos, conversión análoga digital comunicación serial entre otras), soportadas en el proyecto *Aixt*, escritas en *C* y enfocadas para el microcontrolador RP2040.
Archivo de configuración	Dado que se necesita un soporte para el transcompilador del lenguaje *Aixt* a *C* a la serie de microcontroladores RP2040, en esta etapa se escribirá un archivo de configuración en formato YAML, que será usado como insumo por el transcompilador.
Documentación software libre	Tanto la arquitectura del hardware de los microcontroladores RP2040, la interfaz de programación de aplicaciones y la serialización, necesitan tener un soporte escrito que las expliquen, identifiquen y caractericen.

Tabla 1: Plan de trabajo

CRONOGRAMA

Mes	**1**	**2**	**3**	**4**	**5**
Diseño de la arquitectura hardware					
Desarrollo de la API					
Archivo de configuración para la transcompilación					
Documentación software libre					

Tabla 2: Cronograma

3. MARCO TEÓRICO

En primera medida se realizó una investigación sobre el microcontrolador asignado en este caso, el RP2040 con la tarjeta Raspberry Pi Pico, con el fin de conocer sus características físicas, su software y hardware, sus capacidades, memoria, velocidad, entre otros.

Raspberry Pi Pico es la primera placa de clase microcontrolador de Raspberry Pi. Construido sobre la plataforma de silicio RP2040, Pico aporta características de alto rendimiento, bajo costo y facilidad de uso en el espacio de los microcontroladores. (Raspberry Pi, s.f.)

Especificaciones:

Factor de forma: 21 mm x 51 mm

CPU: Dual-core Arm Cortex-M0 + @133 MHz

Memoria: 264 KB on chip SRAM; 2 MB on board QSPI flash

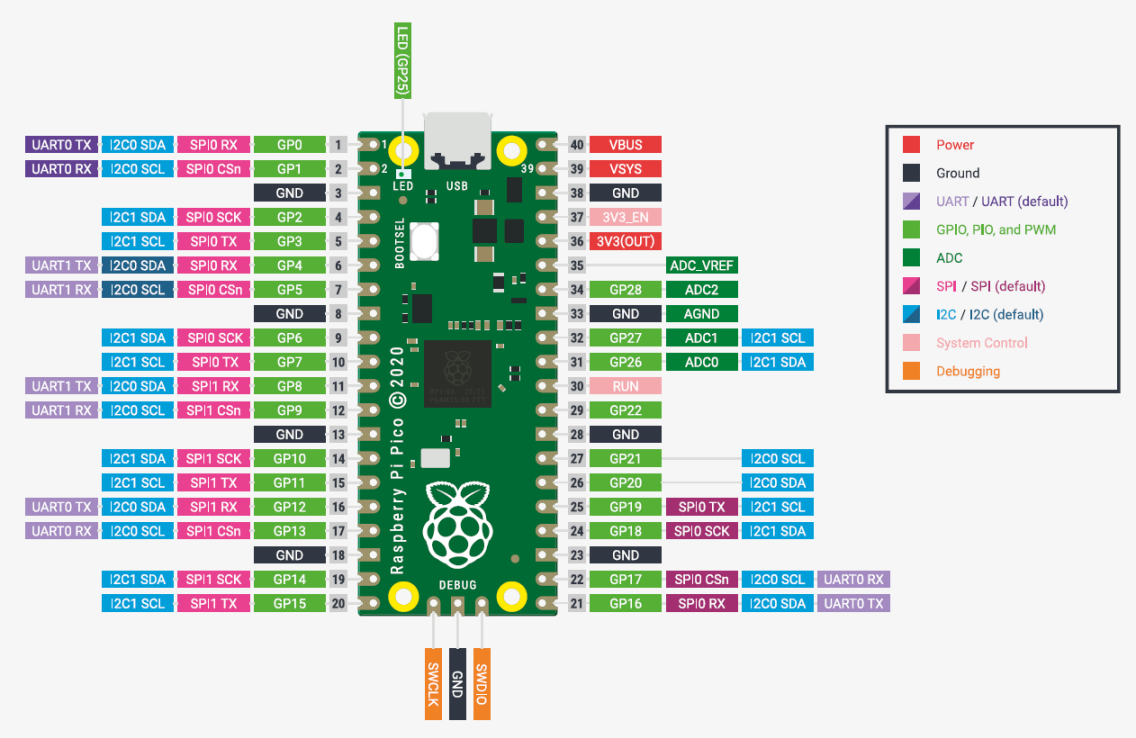
Interfaz: 26 pines GPIO, incluyendo 3 entradas análogas

Periféricos:
- 2 × UART
- 2 × controladores SPI
- 2 × controladores I2C
- 16 × canales PWM
- 1 × controlador USB 1.1 y PHY, con soporte para host y dispositivo
- 8 × máquinas de estado PIO

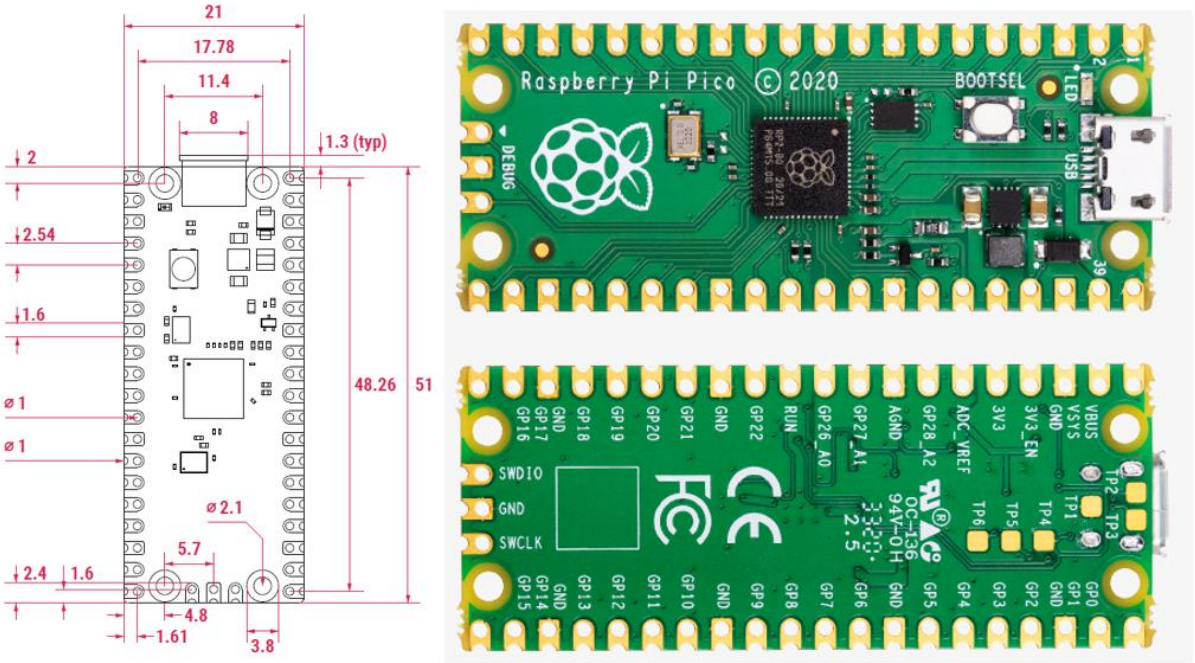
Potencia de entrada: 1.8 – 5.5 V DC

Temperatura de operación: -20 °C a + 85°C

Vida útil de producción: Raspberry Pi Pico permanecerá en producción hasta al menos enero 2028



ESQUEMA 1



ESQUEMA 2

4. DESARROLLO DE PASANTÍA

Objetivo específico	Actividades (Cronograma)	Resultados
Dar soporte a un lenguaje moderno basado en V para ser transcompilado de manera efectiva en lenguaje C para microcontroladores de la serie RP2040.	*Se programan los códigos iniciales y las bases para desarrollar sobre estos las definiciones en lenguaje V.*	*Capítulo 4.1 y 4.2.1*
	Crear los módulos que contengan los archivos ".C.V." con las definiciones de las funciones implementadas en lenguaje V.	
Desarrollar un conjunto de funciones básicas en C para microcontroladores de la serie RP2040, que pueda ser usado desde un lenguaje basado en V luego de su transcompilado.	*Generar los archivos de configuración, diccionarios, módulos y demás estructuras para la transcompilación de los códigos.*	*Capítulo 4.2.2 y 4.2.3*
	Generar los archivos ".V" con los códigos de la API en lenguaje V, importando las librerías y módulos creados para su transcompilación.	
	Transcompilar los códigos en V, y verificar su correcto funcionamiento en el software "Arduino IDE" de manera que se evidencie las funciones en ambos lenguajes	

	en las librerías y módulos creados.	
Realizar una documentación que permita a los desarrolladores y personas interesadas entender la estructura, funcionalidades y características del código, lo que a su vez también les permitirá contribuir al proyecto si así lo desean, puesto que se desea que sea de código abierto.	*Generación de informe final y mapa conceptual de la estructura de las carpetas y archivos creados.*	*Capítulo 4.2 y 4.2.3*

Tabla 3: Desarrollo de la pasantía

4.1 DESARROLLO DE FUNCIONES BÁSICAS.

Se desarrollaron los códigos de las funciones básicas con el fin de realizar las primeras interacciones con el microcontrolador, se controlará el encendido de un led de forma autónoma, también controlada usando las entradas digitales del dispositivo, también implementaremos el control analógico usando un potenciómetro que controlará salidas digitales, por ultimo implementaremos un código para PWM y para comunicación UART.

El software elegido para realizar los códigos iniciales es Arduino IDE, ya que es un editor de código versátil, el cual soporta código en C y la Raspberry Pi Pico.

4.1.1. BLINKING

Este código es el primero y más básico de todos pues su único propósito es encender y apagar un led por un tiempo de 1000 milisegundos (1 segundo) cada estado (encendido - apagado). La tarjeta tiene incorporada un led asignado al pin 25, es por esto que para iniciar la interacción con la tarjeta es el código más sencillo a implementar ya que no requiere de elementos adicionales. Sin embargo, es posible implementar el mismo código con elementos adicionales como un led externo y una fuente externa.

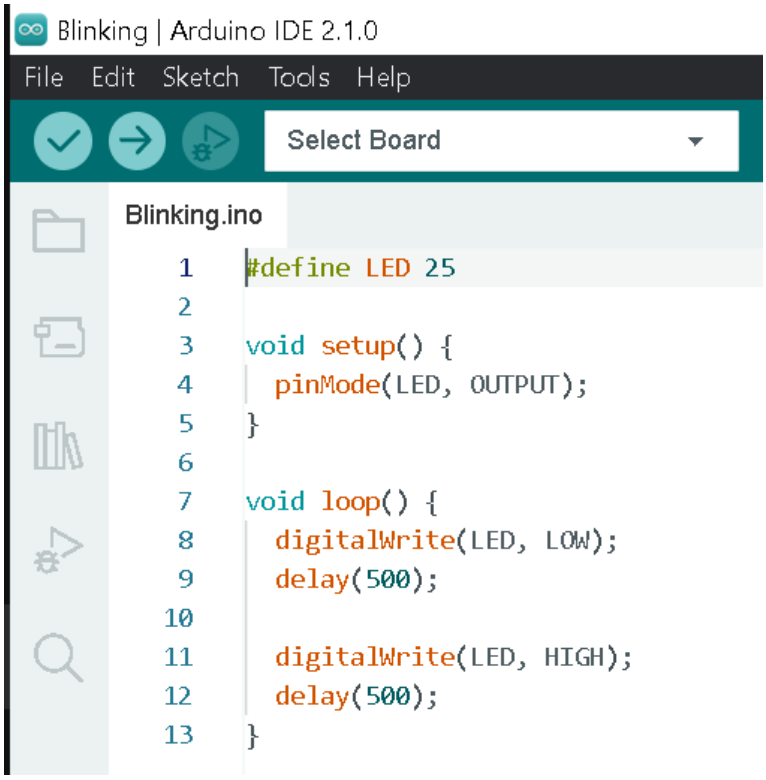


Figura 1: Código de programación parpadeo led integrado

4.1.2. Digital input/output

Para esta parte del proceso ya debemos implementar elementos adicionales pues como su nombre lo indica, el encendido y apagado del led será controlado por medio de un interruptor (entrada digital 15). El propósito es usar este interruptor como un botón, el cual al ser oprimido enciende el led y al oprimirse de nuevo lo apaga.

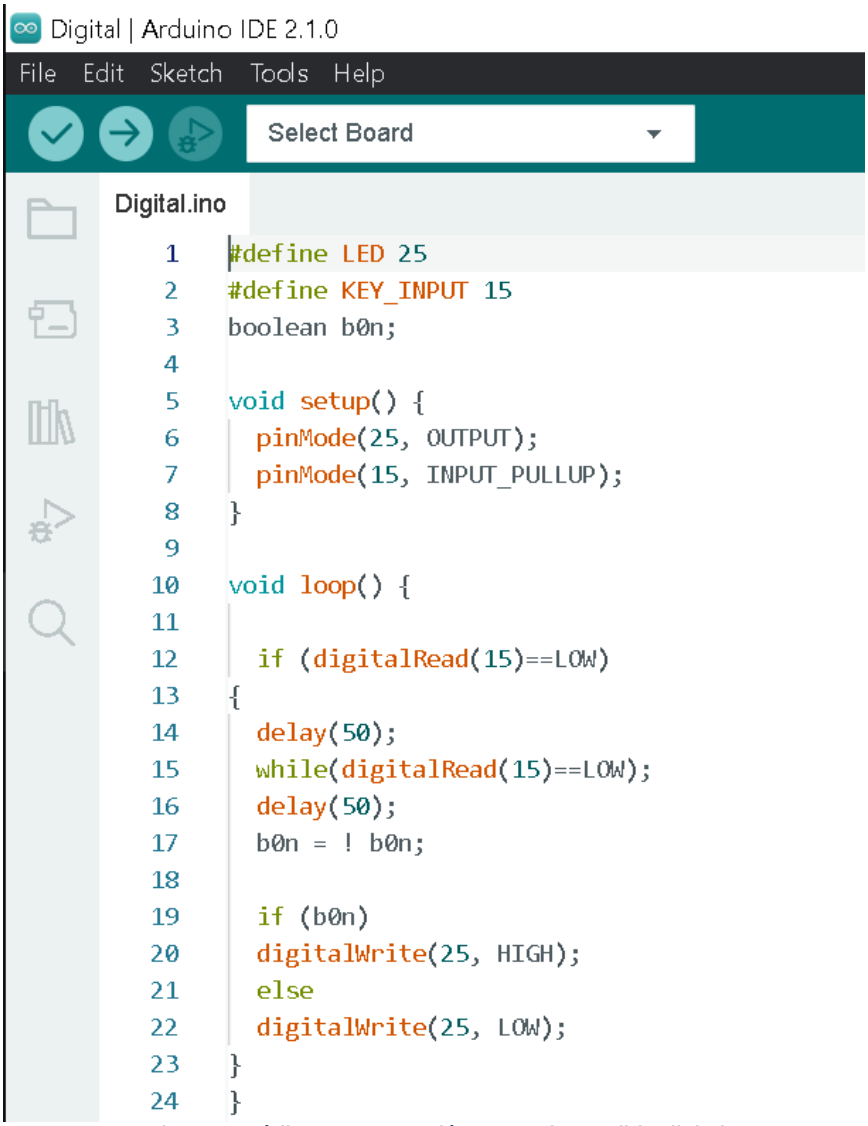


Figura 2 Código programación: Entrada y salida digital

Sintaxis:

pinMode(pin, mode)

Parámetros

pin: Número del pin a configurar

mode: INPUT, OUTPUT, o INPUT_PULLUP.

Ejemplo:

void setup() { pinMode(25, OUTPUT);

pinMode(15, INPUT_PULLUP);

}

4.1.3. Analog Inputs (ADC)

El conversor análogo a digital es uno de los códigos más usados en la aplicación de programación a la electrónica. En este caso lo usaremos para controlar el encendido de 4 leds, conectados a las salidas digitales 13, 14, 15 y 16 a medida que se va girando el potenciómetro, el cual ira conectado a una de las entradas análogas (A1) del Raspberry Pi Pico

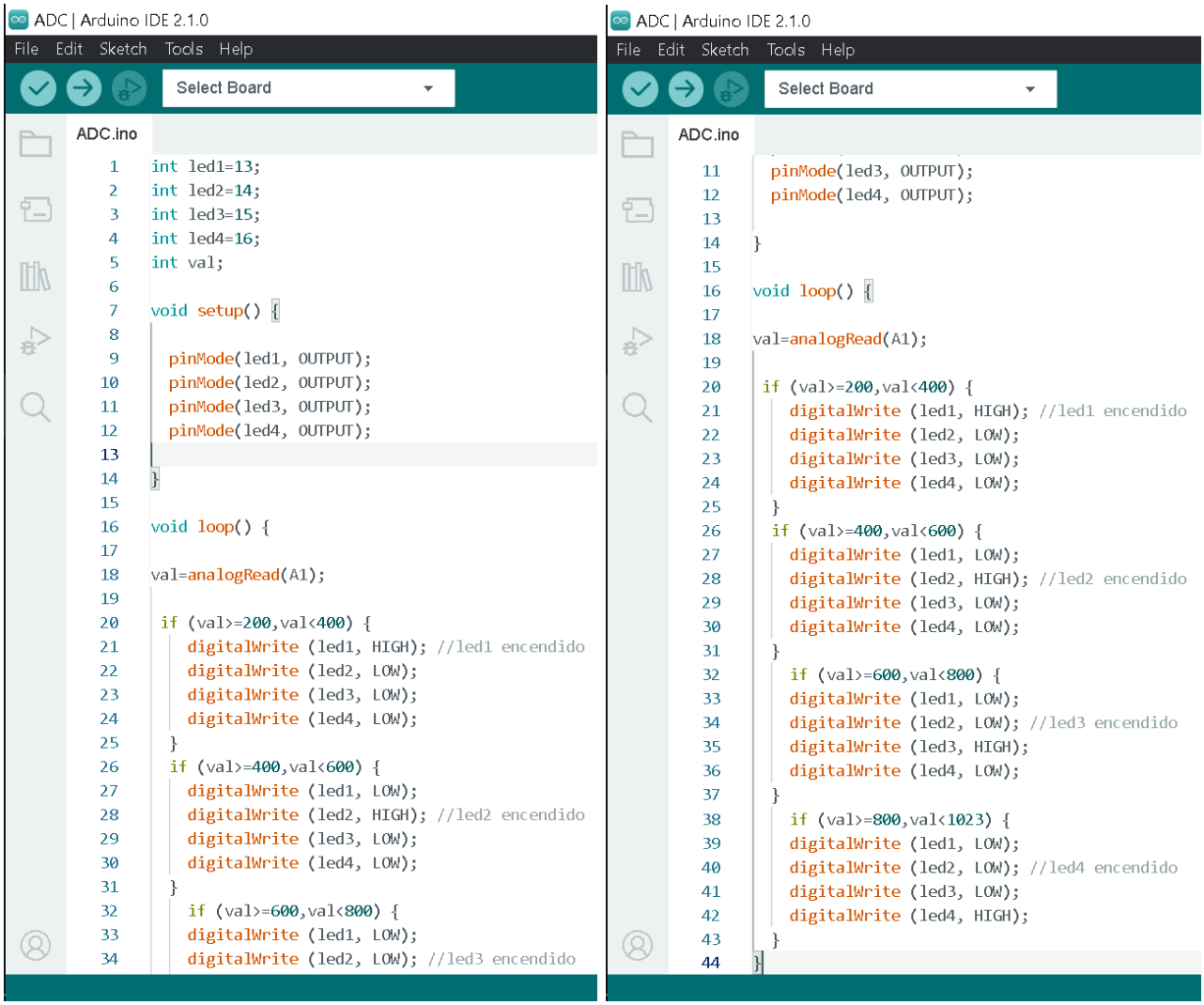


Figura 3: Código programación convertidor análogo a digital

Sintaxis:

analogRead(pin)

Parámetros:

pin: el nombre del pin de entrada analógica para leer

Ejemplo:

```
int val;

void loop() {

val=analogRead(A1);

 }
```

4.1.4. PWM output

La modulacion por ancho de pulso tambien es una de los codigos mas usados en la industria, sobretodo en la electronica de potencia, ya que permite controlar la velocidad o potencia de un dispositivo. Aca lo implementamos para controlar la intensidad de brillo de un led.

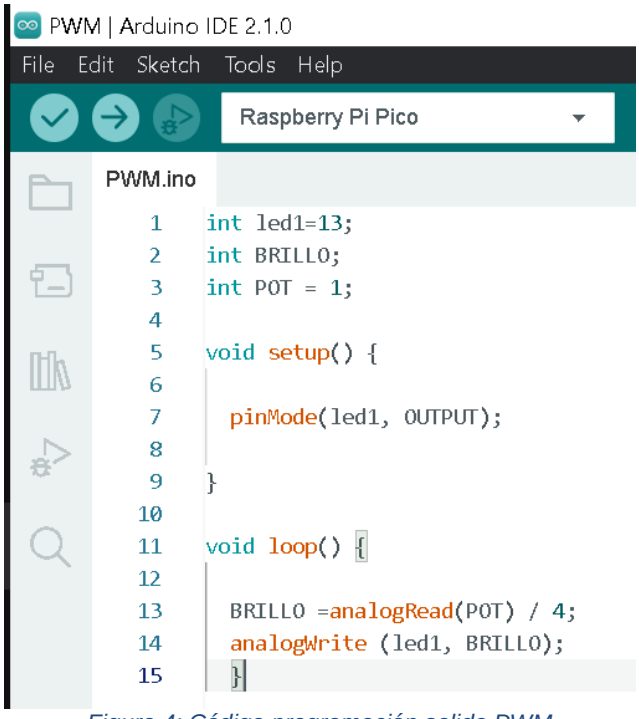


Figura 4: Código programación salida PWM

Sintaxis:
analogWrite(pin, value)

Parámetros:

pin: el nombre del pin a escribir

value: el ciclo de trabajo: entre 0 (siempre apagado) y 255 (siempre encendido)

Ejemplo:

```
int led1=13;
int BRILLO;
int POT = 1;

void setup() {
pinMode(led1, OUTPUT);
}
void loop() {
BRILLO =analogRead(POT) / 4;
analogWrite (led1, BRILLO);
}
```

4.1.5. Comunicación por puerto serial UART

En este código usamos el monitor del software Arduino IDE con el fin de verificar la comunicación serial entre el dispositivo y el PC.



Figura 5: Código programación comunicación serial

Sintaxis:

Serial.begin(speed)

Parámetros:

serial: puerto serial

speed: velocidad de transmisión de datos (bits por segundo - baud)

Ejemplo:

```
void setup () {
Serial.begin(9600);
Serial1.begin(9600);}
```

Sintaxis:

Serial.read()

Serial.available()

Parámetros:

Serial: Puerto serial

Ejemplo:

```
int incomingByte = O; //Para datos seriales entrantes
void setup () {
serial.begin(9600); //Abre el puerto serie con velocidad de 9600 bps
void loop () {
if (Serial.available() > 0);  // Envía datos solo cuando recibes datos
incomingByte = Serial.read();  //Lee los datos entrantes
Serial.print(¨I received: ¨);  //Dice lo que tiene
Serial.println(incomingByte, DEC);  //Dice con un salto de línea el tipo de variable almacenada
```

Sintaxis:

Serial.print(val)

Serial.println(val)

Parámetros:

Serial: Puerto serial

val: el valor a imprimir

Ejemplo:

```
int incomingByte = O; //Para datos seriales entrantes

void setup () {

serial.begin(9600); //Abre el puerto serie con velocidad de 9600 bps

void loop () {

if (Serial.available() > 0);  // Envía datos solo cuando recibes datos

incomingByte = Serial.read();  //Lee los datos entrantes

Serial.print(¨I received: ¨);  //Dice lo que tiene

Serial.println(incomingByte, DEC);  //Dice con un salto de línea el tipo de variable almacenada
```

Funciones de tiempo

Sintaxis:

delayMicroseconds(us)

Parámetros:

us: Número de microsegundos para hacer una pausa

Ejemplo:

```
void setup () {

pinMode(13,OUTPUT);}

void loop () {

digitalWrite (13, HIGH);  //Enciende el led a nivel alto 5V

delayMicroseconds(50);  //Pausa de 50 microsegundos

digitalWrite(13, LOW);  //Apaga el led a nivel bajo 0V
```

```
delayMicroseconds(50);  //Pausa de 50 microsegundos para volver a encender el led}
```

Sintaxis:

delay(ms)

Parámetros:

ms: Número de milisegundos para hacer una pausa

Ejemplo:

```
void setup () {

pinMode(13,OUTPUT);}

void loop () {

digitalWrite (13, HIGH);  //Enciende el led a nivel alto 5V

delay(1000);  //Pausa de 1000 milisegundos (1 segundo)

digitalWrite(13, LOW);  //Apaga el led a nivel bajo 0V

delay(1000);  //Pausa de 1000 milisegundos (1 segundo) para volver a encender el led

 }
```

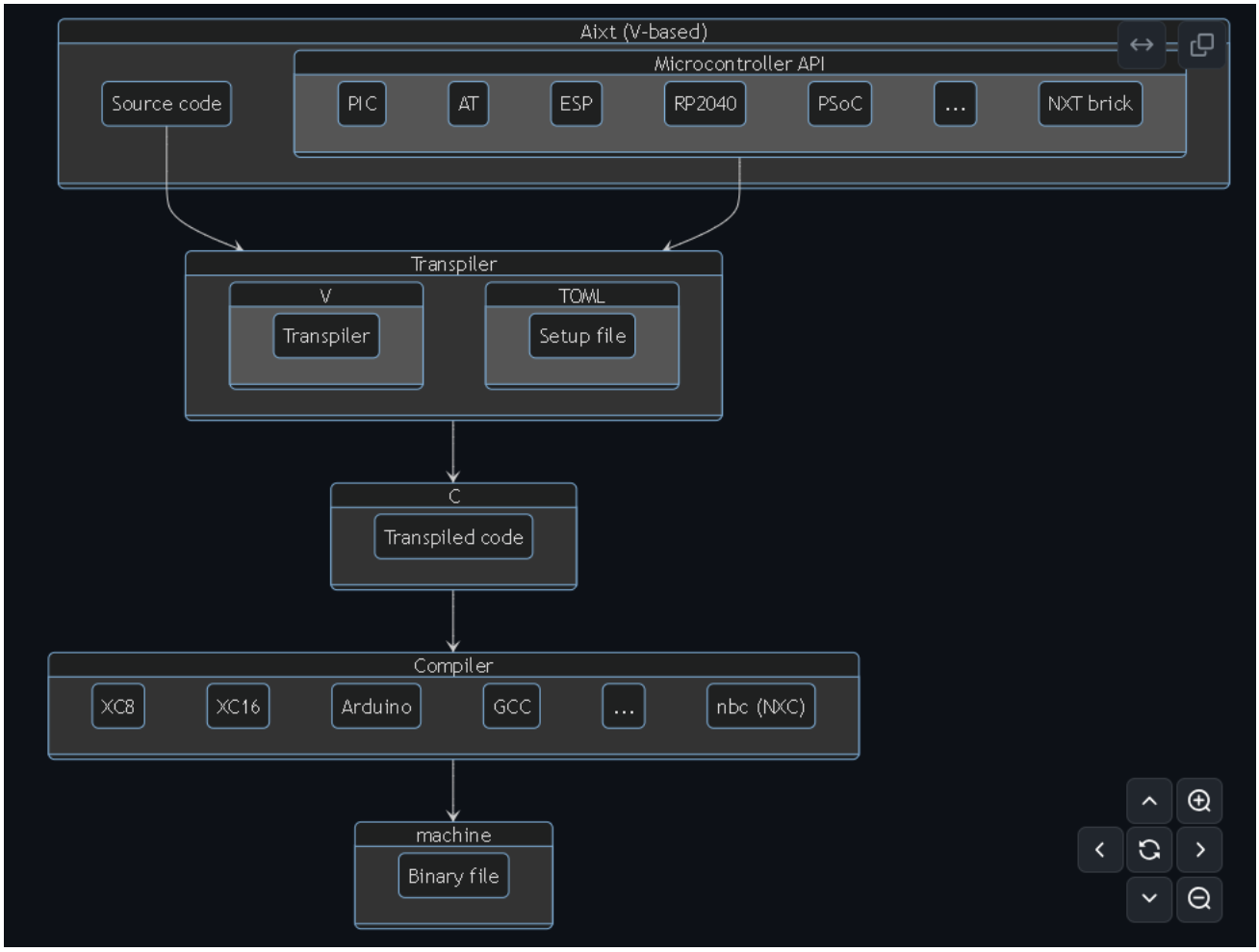
4.2. ESTRUCTURA INTERACTIVA DE AIXT

Luego de interactuar y comprobar los códigos elaborados, verificar sus entradas y salidas y aplicar los conceptos básicos de programación con el microcontrolador, procedemos a entrar en el concepto del proyecto del grupo de investigación. Como se mencionó anteriormente Aixt es un proyecto dirigido a crear un lenguaje de programación versátil, intuitivo y de bajo costo o más asequible.

Aixt es un marco de programación para microcontroladores que utiliza una sintaxis de lenguaje moderno basado en V y capaz de ser utilizado por dispositivos de bajos recursos. Aixt está compuesto por 3 componentes principales:

- El lenguaje de programación Aixt basado en la sintaxis del lenguaje V.
- El Transpiler de Aixt a C, que traduce el código fuente de Aixt a C, para el compilador de C específico de cada microcontrolador.
- La API de Aixt, que facilita la programación estandarizando las funciones de configuración y E/S. (Santa, 2022 -2024)

Este diagrama muestra los bloques de Aixt y sus interacciones:



Esquema 3: Diagrama interactivo de Aixt

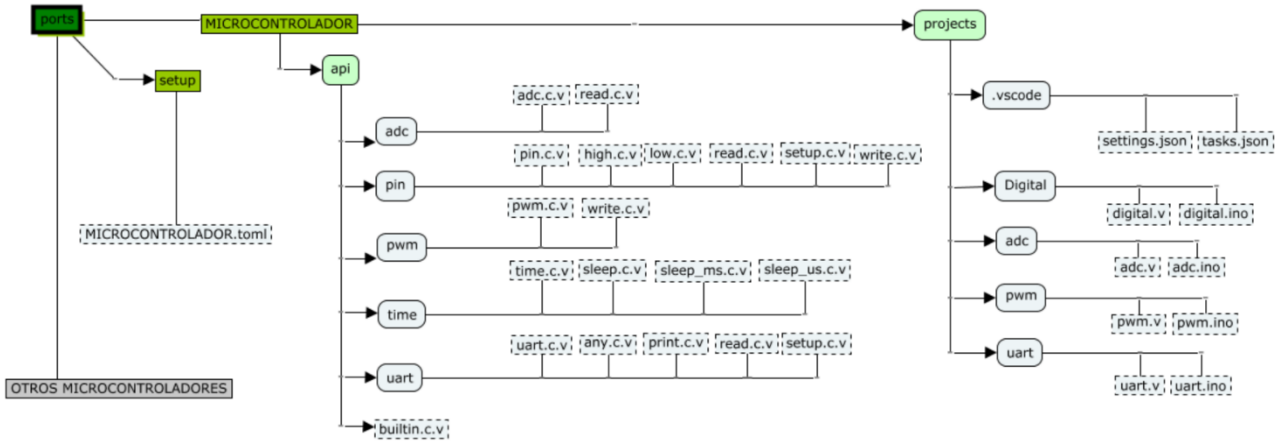
Podemos ver que el código fuente está relacionado con la tarjeta o microcontrolador, la intención de Aixt es abarcar la mayor cantidad de microcontroladores del mercado, con la intención de hacerlo accesible para la mayoría de estudiantes, proyectos académicos e incluso ejecutar proyectos domótica, industriales y darle un uso más comercial en el mercado.

Según el diagrama y luego de elaborar los primeros códigos, estamos en la primera parte del proceso, entonces el paso a seguir es elaborar la parte del transpilador, el cual está compuesto códigos que dan una definición en leguaje V a los códigos inicialmente planteados.

Para generar una estructura clara y que pueda ser entendida por otras personas se presenta un orden de carpetas según los códigos iniciales:

- **Digital input/output**
- **Analog inputs (ADC)**
- **PWM outputs**
- **Serial port (UART)**

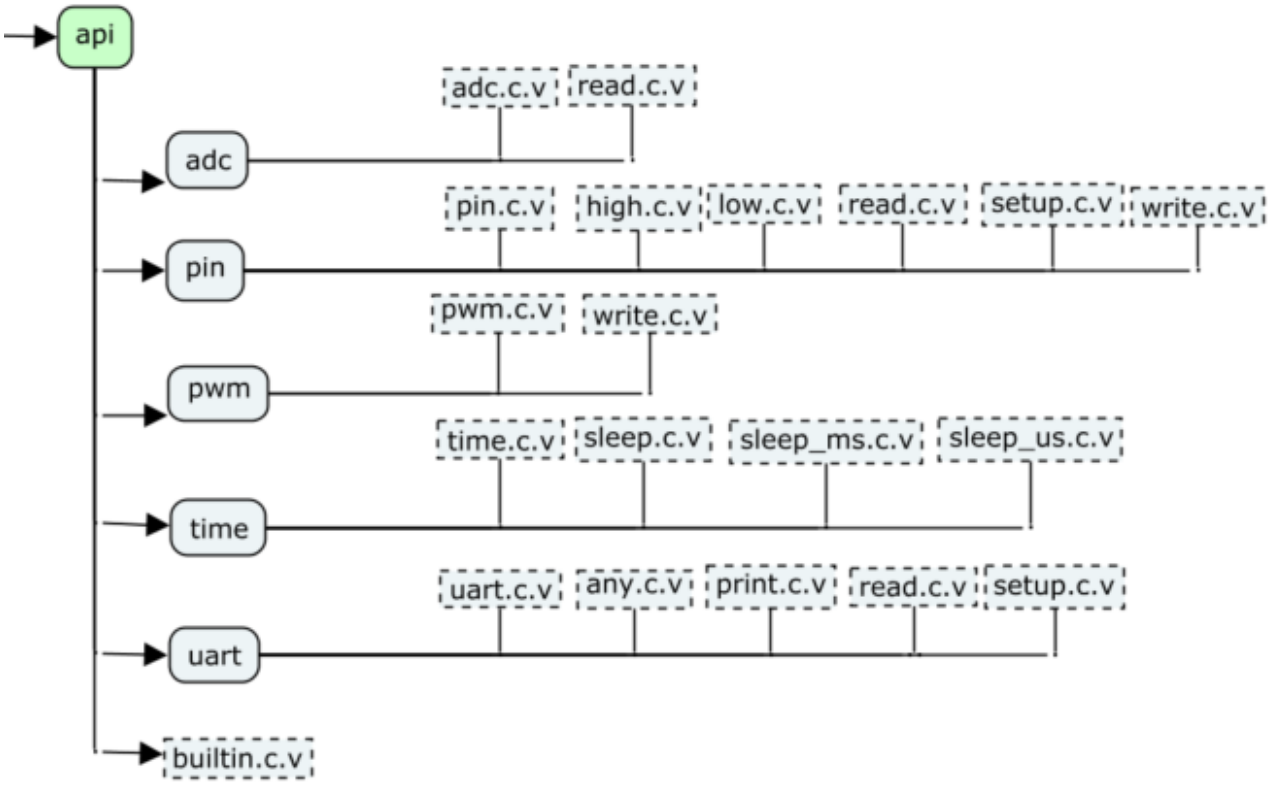
Para esto la estructura definida en las carpetas busca que las funciones estén agrupadas de forma coherente según los códigos elaborados:



Esquema 4: Estructura de carpetas y archivos

4.2.1. Módulos:

Dentro de la carpeta *ports* están incluidos todos los archivos y subcarpetas en las cuales se trabajan los códigos y programas durante el desarrollo de la pasantía. También se encuentra la carpeta *setup* la cual es una carpeta que contiene un archivo *. toml* el cual trabajaremos más adelante.
Como se mencionó anteriormente, iniciamos con las definiciones de las funciones usadas en código V, estas definiciones estarán incluidas en archivos “. c. v”, los cuales están incluidos en subcarpetas dentro de la carpeta “api”, estas subcarpetas serán llamadas “módulos”.



Esquema 5: Estructura de subcarpetas y archivos en api

Cada uno de los módulos contiende funciones del tipo #define, lo cual no indica que podemos crear una librería de funciones en dos lenguajes, los cuales serán usados para transcompilar los archivos de la API en lenguaje V.

Modulo pin:

pin.c.v

pin.c.v

ports > Raspberry-Pi > Raspberry-Pi-Pico > api > pin > pin.c.v

```
1  module pin
2
3  #define pin__output     OUTPUT
4  #define pin__input      INPUT
5  #define pin__in_pullup  INPUT_PULLUP
6
7  fn init() {
8
9  }
```

Figura 6: Código en archivo - pin.c.v – modulo pin

setup.c.v



Figura 7: Código en archivo - setup.c.v – modulo pin

high.c.v



Figura 8: Código en archivo - high.c.v – modulo pin

low.c.v



Figura 9: Código en archivo - low.c.v – modulo pin

read.c.v



Figura 10: Código en archivo - read.c.v – modulo pin

write.c.v



Figura 11: Código en archivo - write.c.v – modulo pin

Módulo adc:

adc.c.v

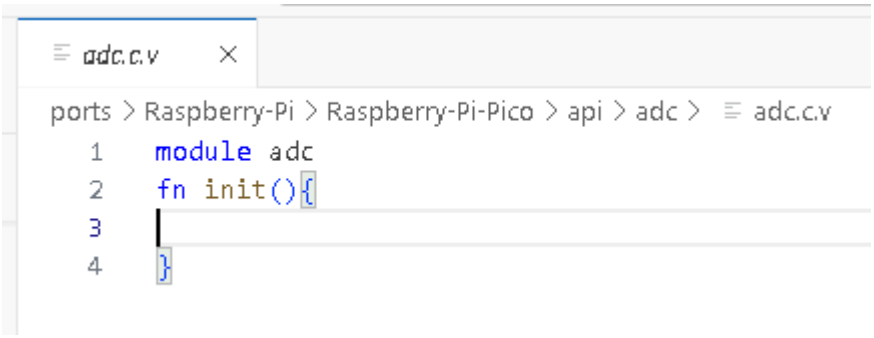


Figura 12: Código en archivo - adc.c.v – módulo adc

read.c.v



Figura 13: Código en archivo - read.c.v – módulo adc

Módulo pwm

pwm.c.v

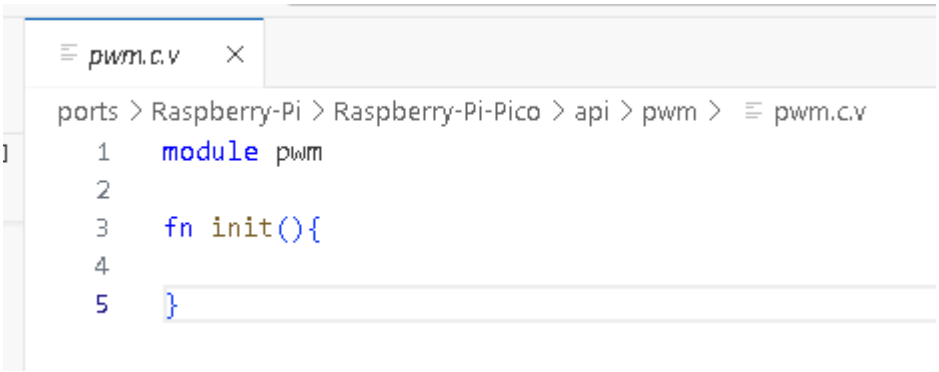


Figura 14: Código en archivo - pwm.c.v – módulo pwm

write.c.v

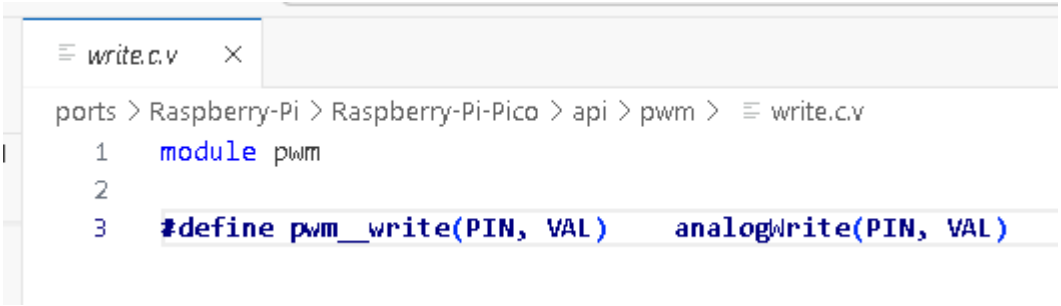


Figura 15: Código en archivo - write.c.v – módulo pwm

Módulo uart:

uart.c.v

uart.c.v

ports > Raspberry-Pi > Raspberry-Pi-Pico > api > uart > uart.c.v

```
1  module uart
2
3  fn init() {
4
5  }
```

Figura 16: Código en archivo - uart.c.v - módulo uart

any.c.v

any.c.v

ports > Raspberry-Pi > Raspberry-Pi-Pico > api > uart > any.c.v

```
1  module uart
2
3
4
5  #define uart__any_0()   Serial.available()
6
7  #define uart__any_1()   Serial1.available()
8
9  #define uart__any_x(UART_NUMBER)    uart__any_ ## UART_NUMBER ## .available()
10
11 // macro with 0 or 1 arguments
12 #define SEL_UART_ANY(_0, _1, MACRO_NAME, ...) MACRO_NAME
13 #define uart__any(...) SEL_UART_ANY(_0 __VA_OPT__(,) __VA_ARGS__, uart__any_x, uart__any_0)(__VA_ARGS__)
```

Figura 17: Código en archivo - any.c.v - módulo uart

setup.c.v

setup.c.v

ports > Raspberry-Pi > Raspberry-Pi-Pico > api > uart > setup.c.v

```
1  module uart
2
3  #define uart__setup(BAUD_RATE)   Serial.begin(BAUD_RATE)
4
5  #define uart__setup_0(BAUD_RATE)                    Serial.begin(BAUD_RATE)
6
7  #define uart__setup_1(BAUD_RATE)                    Serial1.begin(BAUD_RATE)
8
9  #define uart__setup_x(UART_NUMBER, BAUD_RATE)       uart__setup_ ## UART_NUMBER (BAUD_RATE)
10
11 // macro with 1 or 2 arguments
12 #define SEL_UART_SETUP(_1, _2, MACRO_NAME, ...) MACRO_NAME
13 #define uart__setup(...) SEL_UART_SETUP(__VA_ARGS__, uart__setup_x, uart__setup_0)(__VA_ARGS__)
```

Figura 18: Código en archivo - setup.c.v - módulo uart

read.c.v

read.c.v

ports > Raspberry-Pi > Raspberry-Pi-Pico > api > uart > read.c.v

```
1   module uart
2
3
4   #define uart__read_0()  Serial.read()
5
6   #define uart__read_1()  Serial1.read()
7
8   #define uart__read_x(UART_NUMBER)   uart__read_ ## UART_NUMBER ## .read()
9
10  // macro with 0 or 1 arguments
11  #define SEL_UART_READ(_0, _1, MACRO_NAME, ...) MACRO_NAME
12  #define uart__read(...) SEL_UART_READ(_0 __VA_OPT__(,) __VA_ARGS__, uart__read_x, uart__read_0)(__VA_ARGS__
```

Figura 19: Código en archivo - read.c.v - módulo uart

print.c.v

print.c.v

ports > Raspberry-Pi > Raspberry-Pi-Pico > api > uart > print.c.v

```
1   module uart
2
3
4   #define uart__print_0(MESSAGE)  Serial.print(MESSAGE)
5
6   #define uart__print_1(MESSAGE)  Serial1.print(MESSAGE)
7
8   #define uart__print_x(UART_NUMBER, MESSAGE)     uart__print_ ## UART_NUMBER (MESSAGE)
9
10  // macro with 1 or 2 arguments
11  #define SEL_UART_PRINT(_1, _2, MACRO_NAME, ...) MACRO_NAME
12  #define uart__print(...) SEL_UART_PRINT(__VA_ARGS__, uart__print_x, uart__print_0)(__VA_ARGS__)
```

Figura 20: Código en archivo - print.c.v - módulo uart

println.c.v

println.c.v

ports > Raspberry-Pi > Raspberry-Pi-Pico > api > uart > println.c.v

```
1   module uart
2
3
4   #define uart__println_0(MESSAGE)    Serial.println(MESSAGE)
5
6   #define uart__println_1(MESSAGE)    Serial1.println(MESSAGE)
7
8   #define uart__println_x(UART_NUMBER, MESSAGE)       uart__println_ ## UART_NUMBER (MESSAGE)
9
10  // macro with 1 or 2 arguments
11  #define SEL_UART_PRINTLN(_1, _2, MACRO_NAME, ...) MACRO_NAME
12  #define uart__println(...) SEL_UART_PRINTLN(__VA_ARGS__, uart__println_x, uart__println_0)(__VA_ARGS__)
```

Figura 21: Código en archivo - println.c.v - módulo uart

builtin.c.v

builtin.c.v ●

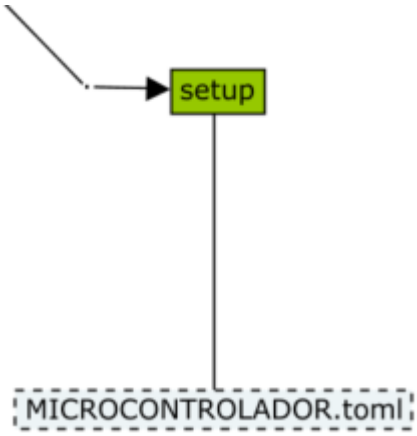
ports > Raspberry-Pi > Raspberry-Pi-Pico > api > builtin.c.v

```
1   // Project Name: Aixt project https://gitlab.com/fermarsan/aixt-project.git
2   // File Name: builtin.c
3   // Date: 2024
4   // License: MIT
5   // Description: Builtin definitions
6
7   module main
8   #define led     gp25
9   enum __pin_names {    // enumerated type for the pin names
10      gp0
11      gp1
12      gp2
13      gp3
14      gp4
15      gp5
16      gp6
17      gp7
18      gp8
19      gp9
20      gp10
21      gp11
22      gp12
23      gp13
24      gp14
25      gp15
26      gp16
27      gp17
28      gp18
29      gp19
30      gp20
31      gp21
32      gp22
33      gp23
34      gp24
35      gp25
36      gp26
37      gp27
38      gp28
39      gp29
40  }
41  fn init() {
42  }
```

Figura 22: Código en archivo - builtin.c.v

Este último no es un módulo ya que no tiene una carpeta y su labor es diferente a los anteriores, en este archivo creamos una lista de los pines y su enumeración según el datasheet del fabricante, esto con la función *enum.* También se le da una definición al pin 25, el cual es el led que está incorporado en la Raspberry Pi Pico.

4.2.2. Setup:



Esquema 6: Estructura en la seccion SETUP

Raspberry-Pi-Pico.toml

Es un archivo creado para que funcione a modo de diccionario para los tipos de dato que soporta el compilador, en este caso, Arduino IDE, con el que finalmente compilaremos los códigos que se generan luego de transcompilar.

Raspberry-Pi-Pico.toml M

ports > setup > Raspberry-Pi-Pico.toml > ...

```
# Project Name = Aixt, https ://github.com/fermarsan/aixt.git
# Author = Camilo Gutierrez
# Date = 2024
# License = MIT
#
# Description = This is the setup file for the Explorer16 (with PIC24)
#               port of Aixt project
port =    "Raspberry-Pi-Pico"
device =  "RP2040"
board =   "Raspberry Pi Pico"
backend = "arduino"

cc_linux =   "xc16"
cc_windows = "C:\\Users\\Anderson\\AppData\\Local\\Programs\\Arduino.exe"
cc_flags =   "-o"

path = "Raspberry-Pi/Raspberry-Pi-Pico"

fixed_size_arrays =  "true"
array_default_len  = 20

fixed_size_strings = "true"
string_default_len = 80

void =  "void"
bool =  "bool"

i8 =    "int8_t"  #"char"
i16 =   "int16_t"
int =   "int32_t"
i64 =   "int64_t"
isize = "int32_t"
int_literal = "int32_t"

u8 =    "uint8_t"
u16 =   "uint16_t"
u32 =   "uint32_t"
u64 =   "uint64_t"
usize = "uint32_t"

f32 = "float"
f64 = "=== NOT SUPPORTED ==="
```

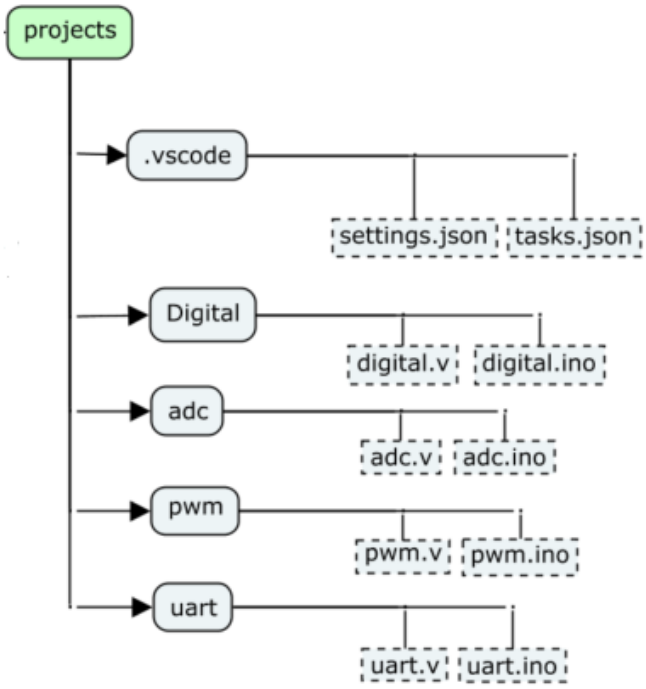
Raspberry-Pi-Pico.toml M

ports > setup > Raspberry-Pi-Pico.toml > [ ] api_modules

```
40
41  f32 = "float"
42  f64 = "=== NOT SUPPORTED ==="
43  float_literal = "float"
44
45  rune =  "char"
46  string = "char []"
47
48  mutex = "=== NOT SUPPORTED ==="
49
50  main_ret_type = "int"
51  main_params =   "void "
52
53  api_modules = [
54      "pin",
55      "adc",
56      "pwm",
57      "uart",
58      "time"
59  ]
60
61  [aliases]
62
```

Figura 23: Código en archivo - Raspberry-Pi-Pico.toml

4.2.3. PROJECTS



Esquema 7: Estructura en sección PROJECTS

La última parte de la estructura en las carpetas y códigos esta en *projects,* acá encontraremos los códigos inicialmente elaborados en C, pero esta vez en lenguaje V, dentro de cada archivo (.c.v)

El *settings.json* es un archivo usado para asociar el compilador la ruta de acceso al *ports* de Raspberry Pi Pico.

.vscode

Settings.json

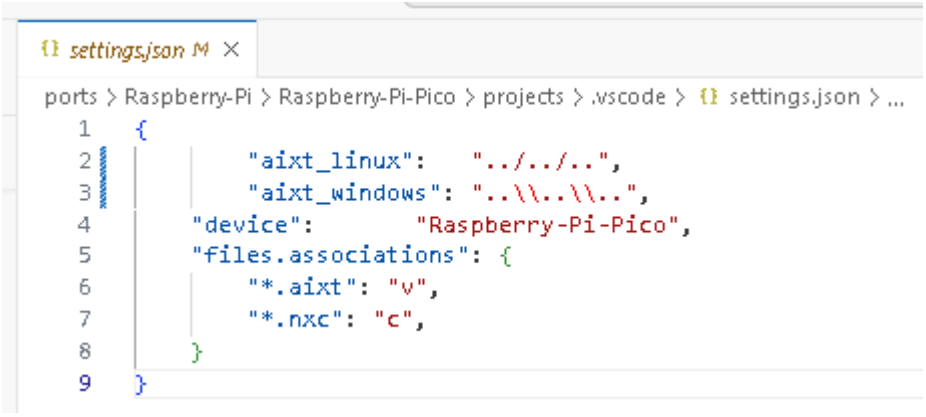


Figura 24: Código en archivo - settings.json

DIGITAL.v

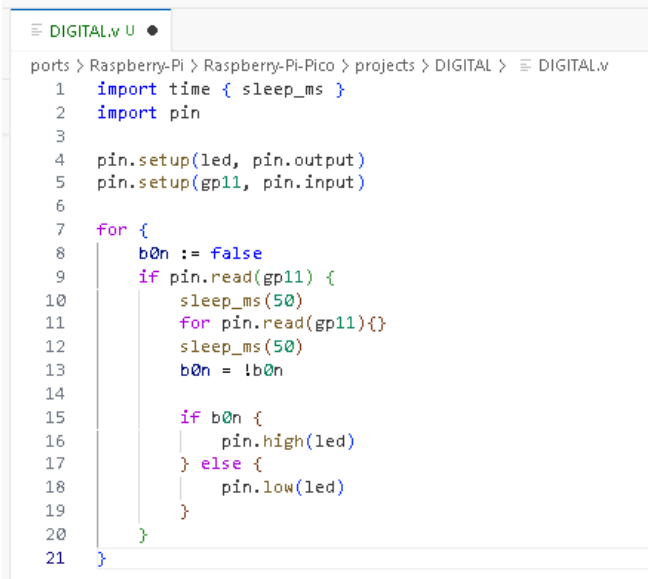


Figura 25: Código en archivo - DIGITAL.v - sección PROJECTS

DIGITAL.ino

DIGITAL.v U ● DIGITAL.ino 6, U ×

ports > Raspberry-Pi > Raspberry-Pi-Pico > projects > DIGITAL > DIGITAL.ino > main___pin_names

```
1   // This Arduino code was automatically generated by Aixt Project
2   // Device = RP2040
3   // Board = Raspberry Pi Pico
4   // Backend = arduino
5
6   #define led     gp25
7   #define time__sleep_ms(MS)    delay(MS)
8   #define pin__output     OUTPUT
9   #define pin__input      INPUT
10  #define pin__in_pullup  INPUT_PULLUP
11  #define pin__high(PIN_NAME)   digitalWrite(PIN_NAME, HIGH)
12  #define pin__low(PIN_NAME)   digitalWrite(PIN_NAME, LOW)
13  #define pin__read(PIN_NAME)   digitalRead(PIN_NAME)
14  #define pin__setup(PIN_NAME, MODE)    pinMode(PIN_NAME, MODE)
15  #define pin__write(PIN_NAME, VALUE)   digitalWrite(PIN_NAME, VALUE)
16
17  void main__init();
18
19  void time__init();
20
21  void pin__init();
22
23  enum main____pin_names {
24      gp0,
25      gp1,
26      gp2,
27      gp3,
28      gp4,
29      gp5,
30      gp6,
31      gp7,
32      gp8,
33      gp9,
34      gp10,
35      gp11,
36      gp12,
37      gp13,
38      gp14,
39      gp15,
40      gp16,
41      gp17,
42      gp18,
```

```
    gp18,
    gp19,
    gp20,
    gp21,
    gp22,
    gp23,
    gp24,
    gp25,
    gp26,
    gp27,
    gp28,
    gp29,
};

void main__init() {
    time__init();
    pin__init();

}

void time__init() {
}

void pin__init() {
}

void setup() {
    main__init();
    pin__setup(led, pin__output);
    pin__setup(gp11, pin__input);
}

void loop() {
    bool b0n = false;
    if(pin__read(gp11)) {
        time__sleep_ms(50);
        while(pin__read(gp11)) {
        }
        time__sleep_ms(50);

        time__sleep_ms(50);
        b0n = !b0n;
        if(b0n) {
            pin__high(led);
        }
        else {
            pin__low(led);
        }
    }
}
```

Figura 26: Código en archivo - DIGITAL.ino - sección PROJECTS

ADC.v

ADC.v

ports > Raspberry-Pi > Raspberry-Pi-Pico > projects > ADC > ADC.v

```
1  import pin
2  import adc
3
4  __global (
5      val = 0
6      )
7  pin.setup(gp13, pin.output)
8  pin.setup(gp14, pin.output)
9  pin.setup(gp15, pin.output)
10 pin.setup(gp16, pin.output)
11
12 for {
13     val = adc.read(gp27)
14
15     if val >= 0 && val < 255 {
16         pin.high(gp13)
17         pin.low(gp14)
18         pin.low(gp15)
19         pin.low(gp16)
20     }
21
22     if val >= 255 && val < 511 {
23         pin.low(gp13)
24         pin.high(gp14)
25         pin.low(gp15)
26         pin.low(gp16)
27     }
28
29     if val >= 511 && val < 767 {
30         pin.low(gp13)
31         pin.low(gp14)
32         pin.high(gp15)
33         pin.low(gp16)
34     }
35
36     if val >= 767 && val < 1023 {
37         pin.low(gp13)
38         pin.low(gp14)
39         pin.low(gp15)
40         pin.high(gp16)
41     }
42 }
```

Figura 27: Código en archivo - ADC.v - sección PROJECTS

ADC.ino

ADC.v ● ADC.ino 9+ ×

ports > Raspberry-Pi > Raspberry-Pi-Pico > projects > ADC > ADC.ino > ...

```
1   // This Arduino code was automatically generated by Aixt Project
2   // Device = RP2040
3   // Board = Raspberry Pi Pico
4   // Backend = arduino
5
6   #define led     gp25
7   #define pin__output     OUTPUT
8   #define pin__input      INPUT
9   #define pin__in_pullup  INPUT_PULLUP
10  #define pin__high(PIN_NAME)   digitalWrite(PIN_NAME, HIGH)
11  #define pin__low(PIN_NAME)   digitalWrite(PIN_NAME, LOW)
12  #define pin__read(PIN_NAME)   digitalRead(PIN_NAME)
13  #define pin__setup(PIN_NAME, MODE)    pinMode(PIN_NAME, MODE)
14  #define pin__write(PIN_NAME, VALUE)   digitalWrite(PIN_NAME, VALUE)
15  #define adc__read(PIN_NAME)   analogRead(PIN_NAME)
16
17  void main__init();
18
19  void pin__init();
20
21  void adc__init();
22
23  int32_t val = 0;
24
25  enum main____pin_names {
26      gp0,
27      gp1,
28      gp2,
29      gp3,
30      gp4,
31      gp5,
32      gp6,
33      gp7,
34      gp8,
35      gp9,
36      gp10,
37      gp11,
38      gp12,
39      gp13,
40      gp14,
41      gp15,
42      gp16,
```

ADC.v ● ADC.ino 9+ ×

ports > Raspberry-Pi > Raspberry-Pi-Pico > projects > ADC > ADC.ino > main___pin_names

```
25  enum main____pin_names {
41      gp15,
42      gp16,
43      gp17,
44      gp18,
45      gp19,
46      gp20,
47      gp21,
48      gp22,
49      gp23,
50      gp24,
51      gp25,
52      gp26,
53      gp27,
54      gp28,
55      gp29,
56  };
57
58  void main__init() {
59      pin__init();
60      adc__init();
61
62  }
63
64  void pin__init() {
65  }
66
67  void adc__init() {
68  }
69
70  void setup() {
71      main__init();
72      pin__setup(gp13, pin__output);
73      pin__setup(gp14, pin__output);
74      pin__setup(gp15, pin__output);
75      pin__setup(gp16, pin__output);
76  }
77
78  void loop() {
79      val = adc__read(gp27);
80      if(val >= 0 && val < 255) {
81          pin__high(gp13);
```

ADC.v ● ADC.ino 9+ ×

ports > Raspberry-Pi > Raspberry-Pi-Pico > projects > ADC > ADC.ino > ...

```
65  }
66
67  void adc__init() {
68  }
69
70  void setup() {
71      main__init();
72      pin__setup(gp13, pin__output);
73      pin__setup(gp14, pin__output);
74      pin__setup(gp15, pin__output);
75      pin__setup(gp16, pin__output);
76  }
77
78  void loop() {
79      val = adc__read(gp27);
80      if(val >= 0 && val < 255) {
81          pin__high(gp13);
82          pin__low(gp14);
83          pin__low(gp15);
84          pin__low(gp16);
85      }
86      if(val >= 255 && val < 511) {
87          pin__low(gp13);
88          pin__high(gp14);
89          pin__low(gp15);
90          pin__low(gp16);
91      }
92      if(val >= 511 && val < 767) {
93          pin__low(gp13);
94          pin__low(gp14);
95          pin__high(gp15);
96          pin__low(gp16);
97      }
98      if(val >= 767 && val < 1023) {
99          pin__low(gp13);
100         pin__low(gp14);
101         pin__low(gp15);
102         pin__high(gp16);
103     }
104 }
105
```

Figura 28: Código en archivo - ADC.ino - sección PROJECTS

PWM.v

PWM.v

ports > Raspberry-Pi > Raspberry-Pi-Pico > projects > PWM > PWM.v

```
1   import pwm
2   import pin
3   import adc
4
5   __global (
6     val = 0
7   )
8
9   pin.setup(gp10, pin.output)
10
11  for {
12
13  val = adc.read(gp28)
14  pwm.write(gp10, val)
15
16  }
```

Figura 29: Código en archivo - PWM.v - sección PROJECTS

PWM.ino

PWM.ino 4

ports > Raspberry-Pi > Raspberry-Pi-Pico > projects > PWM > PWM.ino > ...

```
1   // This Arduino code was automatically generated by Aixt Project
2   // Device = RP2040
3   // Board = Raspberry Pi Pico
4   // Backend = arduino
5
6   #define led     gp25
7   #define pwm__write(PIN, VAL)    analogWrite(PIN, VAL)
8   #define pin__output     OUTPUT
9   #define pin__input      INPUT
10  #define pin__in_pullup  INPUT_PULLUP
11  #define pin__high(PIN_NAME)   digitalWrite(PIN_NAME, HIGH)
12  #define pin__low(PIN_NAME)   digitalWrite(PIN_NAME, LOW)
13  #define pin__read(PIN_NAME)   digitalRead(PIN_NAME)
14  #define pin__setup(PIN_NAME, MODE)    pinMode(PIN_NAME, MODE)
15  #define pin__write(PIN_NAME, VALUE)   digitalWrite(PIN_NAME, VALUE)
16  #define adc__read(PIN_NAME)   analogRead(PIN_NAME)
17
18  void main__init();
19
20  void pwm__init();
21
22  void pin__init();
23
24  void adc__init();
25
26  int32_t val = 0;
27
28  enum main____pin_names {
29      gp0,
30      gp1,
31      gp2,
32      gp3,
33      gp4,
34      gp5,
35      gp6,
36      gp7,
37      gp8,
38      gp9,
39      gp10,
40      gp11,
41      gp12,
42      gp13,
```

PWM.ino 4

ports > Raspberry-Pi > Raspberry-Pi-Pico > projects > PWM > PWM.ino > main___pin_names

```
28  enum main____pin_names {
41      gp12,
42      gp13,
43      gp14,
44      gp15,
45      gp16,
46      gp17,
47      gp18,
48      gp19,
49      gp20,
50      gp21,
51      gp22,
52      gp23,
53      gp24,
54      gp25,
55      gp26,
56      gp27,
57      gp28,
58      gp29,
59  };
60
61  void main__init() {
62      pwm__init();
63      pin__init();
64      adc__init();
65
66  }
67
68  void pwm__init() {
69  }
70
71  void pin__init() {
72  }
73
74  void adc__init() {
75  }
76
77  void setup() {
78      main__init();
79      pin__setup(gp10, pin__output);
80  }
81
```

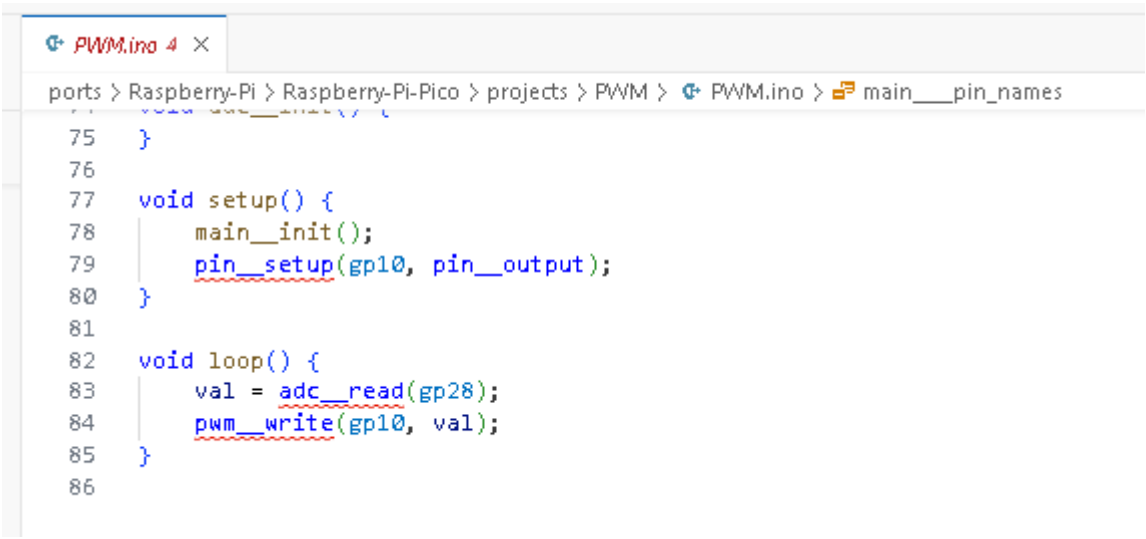


Figura 30: Código en archivo - PWM.ino - sección PROJECTS

UART.v

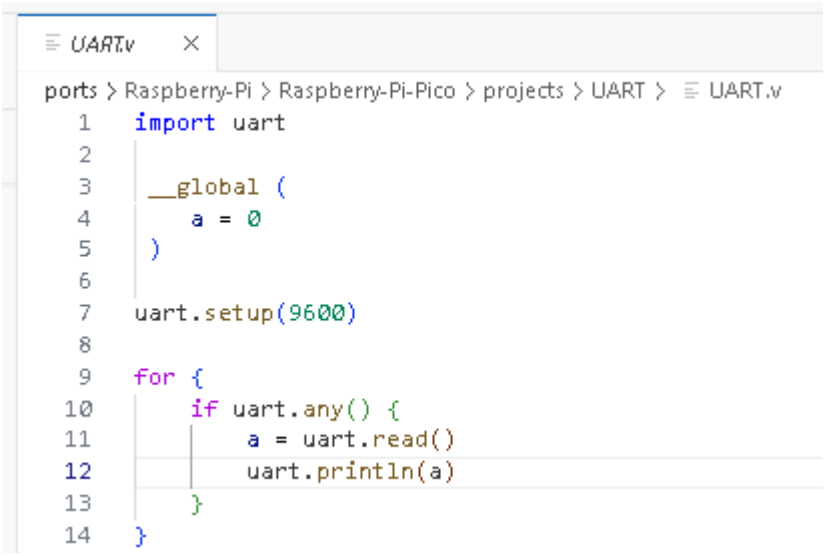


Figura 31: Código en archivo - UART.v - sección PROJECTS

UART.ino

UARTino 3

ports > Raspberry-Pi > Raspberry-Pi-Pico > projects > UART > UART.ino > ...

```
1  // This Arduino code was automatically generated by Aixt Project
2  // Device = RP2040
3  // Board = Raspberry Pi Pico
4  // Backend = arduino
5
6  #define led     gp25
7  #define uart__any_0()   Serial.available()
8  #define uart__any_1()   Serial1.available()
9  #define uart__any_x(UART_NUMBER)    uart__any_ ## UART_NUMBER ## .available()
10 #define SEL_UART_ANY(_0, _1, MACRO_NAME, ...) MACRO_NAME
11 #define uart__any(...) SEL_UART_ANY(_0 __VA_OPT__(,) __VA_ARGS__, uart__any_x, uart__any_0)(__VA_ARGS__)
12 #define uart__print_0(MESSAGE)  Serial.print(MESSAGE)
13 #define uart__print_1(MESSAGE)  Serial1.print(MESSAGE)
14 #define uart__print_x(UART_NUMBER, MESSAGE)     uart__print_ ## UART_NUMBER (MESSAGE)
15 #define SEL_UART_PRINT(_1, _2, MACRO_NAME, ...) MACRO_NAME
16 #define uart__print(...) SEL_UART_PRINT(__VA_ARGS__, uart__print_x, uart__print_0)(__VA_ARGS__)
17 #define uart__println_0(MESSAGE)    Serial.println(MESSAGE)
18 #define uart__println_1(MESSAGE)    Serial1.println(MESSAGE)
19 #define uart__println_x(UART_NUMBER, MESSAGE)       uart__println_ ## UART_NUMBER (MESSAGE)
20 #define SEL_UART_PRINTLN(_1, _2, MACRO_NAME, ...) MACRO_NAME
21 #define uart__println(...) SEL_UART_PRINTLN(__VA_ARGS__, uart__println_x, uart__println_0)(__VA_ARGS__)
22 #define uart__read_0()  Serial.read()
23 #define uart__read_1()  Serial1.read()
24 #define uart__read_x(UART_NUMBER)   uart__read_ ## UART_NUMBER ## .read()
25 #define SEL_UART_READ(_0, _1, MACRO_NAME, ...) MACRO_NAME
26 #define uart__read(...) SEL_UART_READ(_0 __VA_OPT__(,) __VA_ARGS__, uart__read_x, uart__read_0)(__VA_ARGS__)
27 #define uart__setup(BAUD_RATE)   Serial.begin(BAUD_RATE)
28 #define uart__setup_0(BAUD_RATE)                    Serial.begin(BAUD_RATE)
29 #define uart__setup_1(BAUD_RATE)                    Serial1.begin(BAUD_RATE)
30 #define uart__setup_x(UART_NUMBER, BAUD_RATE)       uart__setup_ ## UART_NUMBER (BAUD_RATE)
31 #define SEL_UART_SETUP(_1, _2, MACRO_NAME, ...) MACRO_NAME
32 #define uart__setup(...) SEL_UART_SETUP(__VA_ARGS__, uart__setup_x, uart__setup_0)(__VA_ARGS__)
33
34 void main__init();
35
36 void uart__init();
37
38 int32_t a = 0;
39
40 enum main____pin_names {
41     gp0,
42     gp1,
```

UARTino 3

ports > Raspberry-Pi > Raspberry-Pi-Pico > projects > UART > UART.ino > setup()

```
39
40  enum main____pin_names {
41      gp0,
42      gp1,
43      gp2,
44      gp3,
45      gp4,
46      gp5,
47      gp6,
48      gp7,
49      gp8,
50      gp9,
51      gp10,
52      gp11,
53      gp12,
54      gp13,
55      gp14,
56      gp15,
57      gp16,
58      gp17,
59      gp18,
60      gp19,
61      gp20,
62      gp21,
63      gp22,
64      gp23,
65      gp24,
66      gp25,
67      gp26,
68      gp27,
69      gp28,
70      gp29,
71  };
72
73  void main__init() {
74      uart__init();
75
76  }
77
78  void uart__init() {
79  }
80
```

UART.ino 3

ports > Raspberry-Pi > Raspberry-Pi-Pico > projects > UART > UART.ino > ...

```
78  void uart__init() {
79  }
80
81  void setup() {
82      main__init();
83      uart__setup(9600);
84  }
85
86  void loop() {
87      if(uart__any()) {
88          a = uart__read();
89          uart__println(a);
90      }
91  }
92
```

Figura 32: Código en archivo - UART.ino - sección PROJECTS

5. CONCLUSIONES Y RECOMENDACIONES

Para realizar la transcompilación del lenguaje Aixt que está basado en un lenguaje V, el cual se programó a partir de los módulos creados en la API en donde se definieron las diferentes funciones a utilizar para los respectivos puertos asignados en los programas que se van a implementar para el dispositivo Raspberry Pi Pico, se debe de tener en cuenta que los archivo *builtin.c.v* que se encuentra en la API y los archivos *.toml* que se encuentra en la carpeta *setup* deben de estar sincronizados para poder realizar dicha transcompilación de lo contrario dicha transcompilación no será permitida.

Como se aprecia en la transcompilación, cada lenguaje de programación se llega a la misma finalidad, en este caso el lenguaje C debe llegar al mismo resultado del lenguaje V. Esto con el fin de facilitar el uso del microcontrolador a utilizar.

Contribuyendo a la problemática de la migración de lenguajes de programación, aportando al proyecto Aixt una transcompilación de lenguaje V a C podemos identificar que nuestros dispositivos aún pueden tener una vida útil a pesar de ser algo antiguos, por lo tanto, se puede utilizar la transcompilación en lenguajes distintos al lenguaje V y así utilizar estos microcontroladores como herramientas en futuros proyectos de nuestra universidad.

Para el desarrollo del proyecto fue vital la estructura de las carpetas y los archivos que allí se encuentran, es decir, cada elemento dentro del árbol de procesos tiene una función
importante para la ejecución de los códigos y su correcta transcompilación, al ser varios microcontroladores esto implica que muchos de los archivos necesarios serían

compartidos por diferentes grupos, lo que generó, en algunos casos errores y fallas al momento de transcompilar los códigos de V a C.

Para evitar estas fallas fue necesario replantear en algunas ocasiones la organización de las carpetas de manera que no se vieran afectados los demás proyectos o los códigos de otros microcontroladores.

Es importante realizar un análisis de la estructura del microcontrolador identificando sus entradas, salidas y funciones realizando varias pruebas de funcionamiento, para tener las variables y funciones disponibles que se quieren programar.

Para implementación del microcontrolador Raspberry Pi Pico en el proyecto Aixt fue necesario desarrollar una estructura y lenguaje de programación con el software Visual Studio Code y el IDE de Arduino que permite realizar una transcompilación en el lenguaje moderno Aixt, permite realizar su amplio uso de compatibilidad con diferentes lenguajes de programación en la industria.

6. REFERENCIAS

Fernando Martinez Santa - Fredy Martinez - GRUPO ARMOS. (s.f.). *udistrital.edu.co*. Obtenido de http://www1.udistrital.edu.co:8080/web/unidad-de-investigaciones-de-la-facultad-tecnologica/armos1

Raspberry Pi. (s.f.). *raspberrypi.com*. Obtenido de https://www.raspberrypi.com/products/raspberry-pi-pico/

Santa, F. M. (2022 -2024). *github.com/fermarsan/aixt*. Obtenido de https://github.com/fermarsan/aixt



UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS