PROYECTO FINAL

ELECTRONICA DIGITAL II

Carro Seguidor de luz empleando la Raspberry Pi Pico

1st María del Mar Arbeláez Sandoval.
Dpto. de Ingeniería electrónica y en Telecomunicaciones
Universidad de Antioquia
Medellin, Colombia

2nd Julián Sánchez Ceballos
Dpto. de Ingeniería electrónica y en Telecomunicaciones
Universidad de Antioquia
Medellin, Colombia

***Abstract*—Este informe contiene a detalle apartados sobre el diseño, desarrollo e implementación de un sistema móvil seguidor de luz mediante el uso del microcontrolador RP2040 y los periféricos UART, GPIO, PWM, ADC ya incluidos en el microcontrolador, además se muestran fragmentos del código para la implementación mediante el uso del lenguaje ensamblador en su presentación ThumbARM, esta es una presentación de 16 bits del lenguaje ensamblador para una arquitectura ARM, lo que indica una reducción en el numero de instrucciones y de registros y en las combinaciones de instrucciones posibles. También se muestran detalles técnicos del hardware implementado y las limitaciones del sistema.**

***Index Terms*—Hardware, Memoria, Registros, Periféricos, Microcontrolador.**

I. Introducción

Como eje central del desarrollo del proyecto final se tiene el microprocesador de la empresa Raspberry foundation, RP2040, este microcontrolador cuenta con características como:

- Procesador Dual Cortex M0+ @133MHz
- 264kB de memoria SRAM
- 30 pines de propósito general (GPIO)
- 4 canales ADC de 12 bits
- 2 canales UART
- 8x2 canales PWM

Con estas características el objetivo principal de este proyecto es realizar un carro que mediante el uso de sensores de luz (foto resistencias) conectadas a dos de los canales ADC genere una señal PWM y mueva unos motores que permitan seguir una fuente de luz.

II. Materiales y metodología

Para el diseño e implementación del carro se requiere de los siguientes materiales:

- Dos foto-resistencias.
- Dos motorreductores o motores DC.
- Dos diodos LED.
- Resistencias.
- Un integrado L293D.
- Placa de desarrollo Raspberry Pi Pico.

A continuación se muestra el esquema de conexiones, para todos los elementos previamente mencionados: Como requerimiento del proyecto se agregan dos diodos LED que

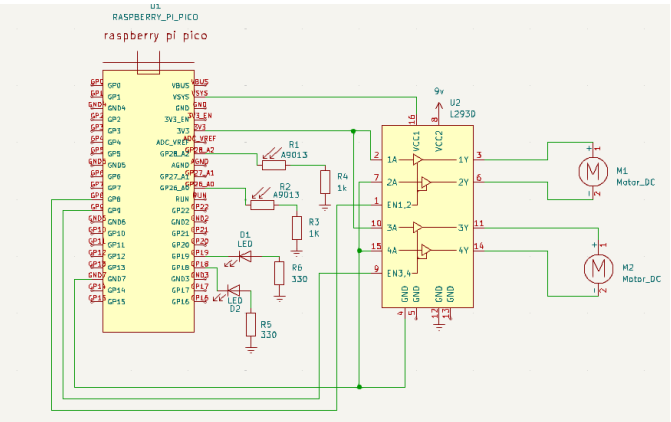

Fig. 1. Diagrama de conexión.

se iluminan dependiendo del motor que se mueva cuando tenga que girar hacia algún sentido, estos LEDs se controlan mediante el periférico GPIO, los sensores son leídos haciendo uso de dos canales ADC, conectados en los GPIO 27-28, por ultimo, la señal que habilita los motores en el controlador L293D son generados en base al valor obtenido de cada canal ADC, un mapeo de variables implementadas en software y haciendo uso del canal PWM para generar un voltaje promedio que mueva los motores dependiendo de la intensidad de la luz aplicada en cada sensor.

Cada periférico se configura previamente a realizar el código principal (main.s) en la entrega del proyecto se adjuntan los archivos correspondientes. Para la configuración de los periféricos ADC y PWM se sigue los siguientes pasos:

ADC:

1) Liberar el Reset del ADC.
2) Configurar el registro CS: Activar el ADC (Enable), seleccionar el canal (AINSEL) y habilitar el start-once (Tomar una sola medida).
3) Configurar el registro DIV: Para una frecuencia de 10KHz el CLK = 12.5.
4) Esperar a que se active el CS.READY para tomar la medida en el canal.

Esta configuración se realiza en el archivo ”adc.s” este archivo contiene dos funciones, una que libera el reset del periférico y otra llamada ”adc_read_asm” esta función es la

encargada de recibir el numero del GPIO donde se encuentra el canal ADC a usar, y retorna la medida encontrada en ese GPIO.

PWM:

1) Liberar el Reset del PWM.
2) Configurar el GPIO como salida PWM.
3) Configurar el registro CLK_DIV: Para una frecuencia de 10KHz se utiliza un valor de 128 en la parte entera y de 0 en la parte decimal.
4) Configuración del registro CC, para que el ciclo de trabajo sea del 50%.
5) Para que el numero de CLK_DIV efectivamente genere una señal de 10KHz este valor se configura a 999.
6) Activar el enable del canal en el registro CHx_CSR.

En el paso 4 se configura un valor para el registro que contiene el valor de COUNTER COMPARE, con este valor se determina el ciclo de dureza de la señal, esto se hace como configuración inicial, más adelante se implementa una función llamada ”Set_cycle_A/B_asm” esta se encarga de recibir el valor de CC y de alterar el registro adecuado para configurar el ciclo de dureza.

En el archivo ”pwm.s” se encuentran 6 funciones: ”pwm_init_asm”, ”pwm_reset_release”, ”setFunctionPWM”, ”pwm_config” y ”Set_cycle_A/B_asm” desde el archivo principal para inicializar el GPIO como una salida pwm, se llama la función ”pwm_init_asm” las funciones para liberar el reset y configurar la función PWM se llaman desde esta función como subrutinas.

Además de estos archivos se agrega un archivo extra llamado ”misc.s” donde se incluyen funciones varias utilizadas a lo largo del código principal como la función de mapeo de variables. Con los periféricos debidamente configurados y los demás archivos debidamente implementados se puede proceder con el funcionamiento del archivo principal. El funcionamiento del archivo main.s, el cual se incluye en la entrega de este proyecto se muestra a continuación con el siguiente diagrama de flujo:

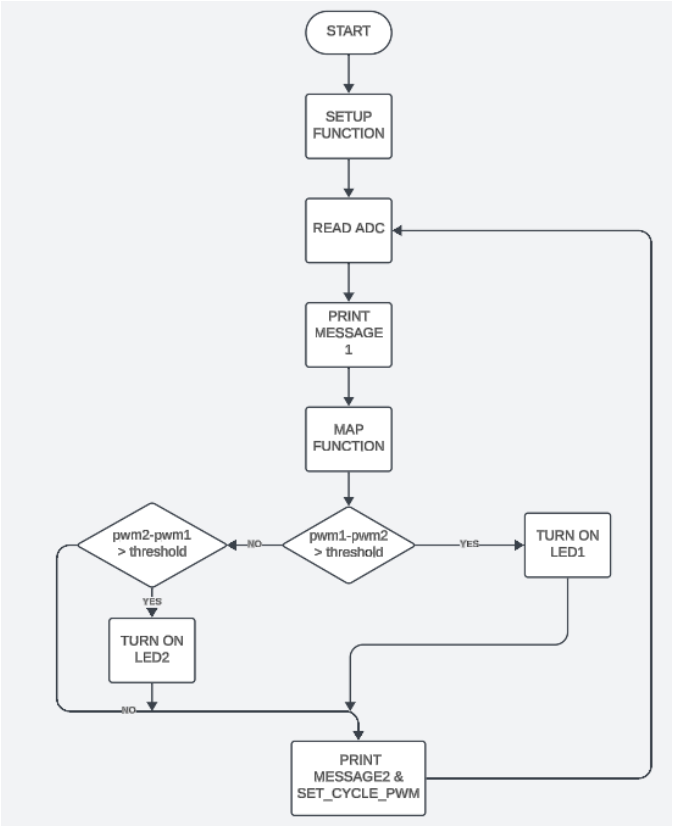

Fig. 2. Diagrama de flujo del programa.

Las funciones de leer el ADC, utilizar la función MAP, encender los LEDs, imprimir mensajes y actualizar los valores del PWM se realiza desde subrutinas implementadas en cada archivo.

III. RESULTADOS:

Como resultado se obtuvo la implementación de un carro que dependiendo de las entradas de los sensores leídos en los canales ADC implementados se imprime un ciclo de dureza en los canales PWM los cuales indican la indican la dirección de movimiento del carro y la velocidad. Sin embargo por errores en la elección de los motores utilizados se pudo observar una diferencia en los motores y por tanto el funcionamiento, aunque correcto y bien implementado, no fue el deseado por motivos de implementación de hardware. El montaje final es posible observarse en la siguiente imagen:

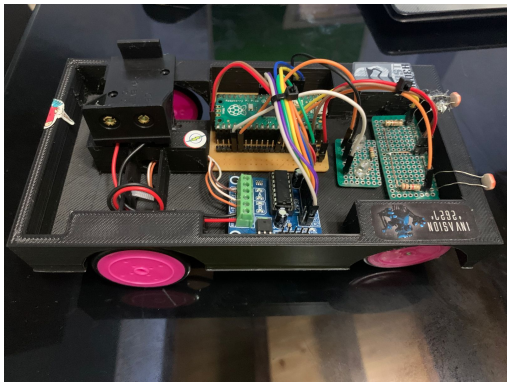

Fig. 3. Montaje final del carro.

IV. Conclusiones

1) Durante la implementación del proyecto se tuvo contacto con el lenguaje ThumbArm el cual representa una versión con muchas más limitaciones respecto a lo que representa propiamente el lenguaje ensamblador en ARM v7 (versión mayormente utilizada durante el transcurso del curso), esto implicó una consulta extra sobre el conjunto de instrucciones y las posibilidades con las que se debe trabajar para el uso de este lenguaje dentro del proyecto.
2) Como se mencionó en reportes anteriores el lenguaje ensamblador permite una programación muy cercana al procesador, incitando al desarrollador conocer de manera más profunda las características del procesador, sus periféricos, el manejo de memoria y registros en este tipo de sistemas, para el uso de este tipo de lenguajes de bajo nivel en sistemas embebidos como lo es la Raspberry Pi Pico implica una habilidad de gran valor en aplicaciones que requieren una optimización de memoria y de eficiencia en el funcionamiento, esto debido a que los kits de desarrollo que usualmente se utilizan para la programación de sistemas embebidos realizan operaciones de más que hacen que el sistema consuma más potencia o energía de la que realmente se requiere debido a que el desarrollador programa de modo que se hagan las tareas estrictamente necesarias.
3) La elección y compra de los elementos son de vital importancia de este tipo de proyectos, la mala elección de uno de los motores impidió el adecuado y deseado funcionamiento del carro, a pesar del correcto funcionamiento de la implementación en software.

References

[1] Raspberry foundation, "Raspberry pi pico datasheet. An RP2040-based microcontroller board," 2020.

[2] R. pi pico, *RP2040 Datasheet. A microcontroller by raspberry*. England: Raspberry foundation, 2020.

[3] R. foundation, *Raspberry pi pico C/C++ SDK. Libraries and tools for C/C++ development on RP2040 microcontrollers.* Raspberry foundation, 2020.