Mi experiencia

juandiazainscough@gmail.com

https://github.com/juandiazainscough/FrameWork-Soldadoras/tree/main

Ingeniería inversa y PIC16F

Ante la problemática contada en “README”, mi jefe estaba en busca de un programador que logre encontrar una solución. Me ofrecí, aunque mi única experiencia era lo que aprendí en la materia “Sistemas Embebidos” en el secundario.

Sabía que no iba a ser posible simplemente extraer el firmware del STM8 original así que, como tenía experiencia con PIC16F intenté desarrollar una placa de cero con él.

Primero hice el trabajo de ingeniería inversa midiendo cada señal y comportamiento de cada pin.

Probé con una soldadora sana activando y desactivando funciones (HotStart, ArcForce, AntiStick, VRD, TIG) e iba comprobando con osciloscopio la variación de cada pin.

Finalmente logré descubrir como el STM8 leía y controlaba cada función. Con eso anotado, me propuse a hacer lo mismo, pero con mi PIC.

El display original usa conexión paralela, pero nosotros contábamos con uno I2C así que me adapté a ello.

Lo primero que hice fue diseñar la librería SSD1306.c para poder escribir caracteres en el display. Todo esto lo hice con MPLAB y Proteus, sin conexiones físicas aún.

Me encontré con una problemática al intentar compilar y era que controlar un display monocromático de 128x64 píxeles con un único buffer requeria 1kB de RAM pero el PIC16F cuenta únicamente con 256 bytes de RAM. Ante esto decidí dividir la pantalla en pequeños bloques e ir moviéndome por ellos mediante una función SetCursor().

Logré hacer que se generara el control de PWM controlado por potenciómetro (no por encoder como la original) y el menú pero noté que la resolución y el tiempo que tardaba en imprimir el display no era apto para comercialización.

Paso STM32

Entendido esto, dejé de lado el PIC y me adentré con STM. Investigando encontré la Bluepill (STM32F103C8T6) que ya era apta para protoboard por lo que era un buen punto de partida. Muy superior al PIC con una RAM de 20kB.

Al principio me costó acostumbrarme a la inicialización de los periféricos y al IDE pero finalmente me di cuenta qué era mucho mejor y más cómodo que MPLAB y el PIC.

Comencé a avanzar más rápidamente ya con pruebas en protoboard y testeos con la placa real.

Lo primero que hice fue diseñar el menú completo utilizando una máquina de estados.

Luego con control por botones podía variar el valor de corriente (A) que mostraba el display.

Control de PWM

Para poder controlar la generación del PWM que va a los IGBTs, el STM8 mandaba una tensión entre 0V y 3V3 al pin COMP del SG3525 (generador de PWM) de la placa de control. Esto era fácil recrear con un potenciómetro, pero no así con botones y, posteriormente, con encoder.

El STM32F103C8T6 carece de DAC (Digital to Analogic Conversor) por lo que no podía generar una tensión entre 0V y 3V3 como hacía el STM8 original.

Ante esto, utilicé la función “PWM generator” del timer de mi STM32. Esto generaba un pulso con un dutycycle que podía variarse según el valor de corriente (A) en mi código.

Mediante un filtro pasabajos RC convertí ese PWM en una señal continua. El dutycycle define la tensión que debe salir. Por ejemplo con un dutycycle a 100% la tensión de salida del filtro es 3V3 pero al 50% es 1V6.

Esto fue probado directamente con una Iron 300 Black conectándole mi protoboard al pin correspondiente y, efectivamente, fue posible controlar la corriente de soldadura.

Encoder

Luego, para reemplazar los botones por un encoder creé una función que leyera el sentido de giro del encoder. Hubo que hacer mucho énfasis en el antirrebote tanto por hardware como por software ya que era muy ruidoso y se hacían muchas falsas detecciones. Se hizo una función que aumentara el valor cada vez más rápido si se seguía girando el encoder de manera ininterrumpida, como hace la soldadora original.

Deteccion de Arco

Para las funciones HotStart, ArcForce y ANtiStick fue necesario trabajar con la detección de tensión de arco. El pin 9 de la placa de control entrega una tensión entre 0V y 5V según la tensión real en el arco (60V = 5V = sin soldar; 0V = 0V = electrodo pegado).

Ésta tensión no podía ser leída directamente por el STM32 ya que supera los 3V3. Usar un regulador de 3V3 tampoco era una opción ya que estos mantienen la tensión estable. Por lo tanto la mejor solución fue hacer un divisor resistivo que entregara la tensión equivalente pero en el rango de 0V a 3V3.

Mediante la función DetectWelding() se lee este Pin para saber el estado del arco. Si no se está soldando (pin = 3V3) al detectarse un cambio en la tensión (pin < 3V3) debía activarse HotStart por un segundo (el tiempo es medido utilizando funciones no bloqueantes, ya que es necesario que el programa siga corriendo). Una vez utilizado el HotStart, este no debe volver a activarse hasta que no se corte el arco por completo y se vuelva a soldar.

HotStart lo que hace es aumentar la corriente (el valor de A en mi código) al incio del arco para facilitar el encendido del electrodo que suele ser un problemático.

Si durante la soldadura, el arco caía mucho (por ejemplo pin <2V) debía activarse el ArcForce hasta que el arco vuelva a estabilizarse (pin > 2V).

ArcForce lo que hace es aumentar la corriente para evitar que el electrodo se pegue a la pieza.

ArcForce no debe activarse cuando está activado HotStart porque sumaría corrientes y esto no tendría sentido.

Noté que la soldadora original, si bien mostraba la función “AntiStick” en el menú del display, realmente no contaba con esta función. Así que la agregué utilizando la función DetectWelding().

Cuando el arco es muy bajo, incluso con ArcForce activado (ejemplo pin < 0V5), quiere decir que el electrodo se pegó a la pieza. Si la soldadora sigue entregando corriente en estas situaciones, complica aún más despegar el electrodo. Justamente está función corta toda generación hasta que se despegue el electrodo.

Las funciones ArcForce y HotStart se manejan con porcentajes (HotStart de 0% a 100% y ArcForce de 0% a 50%) que son seleccionados manualmente por el usuario.

También se cuenta con la posibilidad de seleccionar un diámetro de electrodo especifico con valores ya predeterminados de corriente, HotStart y ArcForce.

Control de generación

Para la función VRD, que se encarga de reducir la tensión entre bornes cuando la soldadora está en reposo por seguridad del usuario, el pin 8 que normalmente está en 5V, puede ser forzado a GND poniéndolo a 0V.

Si este pin está en 5V, la soldadora genera 60V de tensión en vacío, si, en cambio, está en 0V, genera 24V.

Para ello se colocó un transistor NPN con la base controlada por el STM32 que controla la tensión del pin.

En VRD, la tensión de vacío debe mantenerse en 24V hasta que se detecte que se quiere soldar (electrodo toca la pieza), en ese momento vuelve nuevamente a 60V y DetectWelding() funciona con normalidad.

Este mismo pin es utilizado para la función TIG donde el arco de soldadura debe ser en todo momento 24V o dañaría el electrodo de tungsteno.

Cabe destacar, que cuando se selecciona la función TIG, las funciones HotStart, ArcForce y AntiStick se desactivan ya que carecen de sentido en el proceso TIG.

Cuando se está en modo TIG, el menú del display cambia ya que no tendría sentido recorrer submenús de funciones de MMA.

Por último se agregó la función de detección térmica leyendo el pin 3 conectado a un sensor de temperatura. Leyendo este pin es posible saber si el equipo está en la temperatura adecuada de trabajo. En 5V = Temp Ok; en 0V = OverHeat. También se utilizó un divisor resistivo, pero esta vez para ahorrar costo, ya que colocar un AMS1117 era innecesario.

Si se detecta Overheat, se detiene la generación y se muestra un texto de aviso en display impidiendo la soldadura hasta que se vuelva a una temperatura adecuada.

Primer Testeo

Todo esto fue probado en la soldadora original mediante la conexión de una protoboard.

Se notó que el equipo tenía un comportamiento errático con las funciones HotStart y ArcForce.

Por momentos HotStart se mantenía activo por más de 1 segundo e incluso no se desactivaba.

Midiendo con osciloscopio en el pin 9 (el que es leído por DetectWelding() mediante el ADC del STM32) se pudo observar como el ruido electromagnético generado por el arco de soldadura interfería con esta señal induciendo picos incluso superiores a 3V3 (lo que puede ser la causa por lo que se queman estos STM8 en la máquina original).

Al detectar esos picos, el ADC entendía que todavía no se había comenzado a soldar, por lo que mantenía preparado el HotStart.

Filtros

Ante esto, era necesario desarrollar un filtro que filtrara todo ese ruido de múltiples frecuencias y solo dejara la continua que necesitaba leer el ADC.

Se probó con filtros pasivos RLC de hasta 3er Orden y aunque se lograron buenos resultados, aún así, esporádicamente, algún ruido “confundía” al ADC.

Por ello, se intentó con un filtro activo de 2do orden Sallen-Key con una frecuencia de 8Hz que debería filtrar prácticamente cualquier frecuencia generada por el arco.

Para esto se usaron capacitores de 1uF X7R y un LM358 de alta disponibilidad en el taller ya que es muy utilizado en las soldadoras.

Este operacional cuenta con 2 op. Amp., perfectos para el filtro deseado.

Usando este filtro, incluso con cables largos y en una protoboard el ruido prácticamente desapareció permitiendo una lectura del ADC casi perfecta.

Pero para mayor precisión, era necesario reducir la distancia de los cables y evitar loops y vueltas innecesarias. Esto se tuvo en cuenta en el diseño posterior de la PCB.

Diseño de PCB

Una vez testeado todo en protoboard se procedió al diseño de una placa con la intención de hacerla casera con la utilización de Percloruro Férrico.

El PCB fue diseñado pensando en escalabilidad y practicidad en caso necesario de necesitar cambiar el STM32 o reprogramación. Para ello se colocaron headers hembra para poder encastrar la bluepill sin necesidad de soldadura.

Algunas de las características de la PCB son:

- El filtro pasabajos inmediatamente próximo a la entrada del ADC para evitar ruido.
- Cuenta con un AMS1117 para poder convertir los 5V entregados por la fuente de la soldadora en 3V3.
- Encoder con antirrebote por hardware.
- Display SSD1306 con pull-ups para I2C.
- Divisores resistivos para lectura de pines sin riesgo.
- Plano de masa.
- Transistor para control de lógica TIG y VRD.
- Led “HeartBeat” para control.
- Filtro pasabajos para control de PWM.
- Capacitores de filtro entre VCC y GND

Mi aprendizaje

Este proyecto representa mi primera experiencia profesional en el mundo embebido. Comencé casi de cero, únicamente con lo aprendido en el secundario, y con un objetivo.

Si bien no sabía prácticamente nada al inicio, con resiliencia, paciencia y ayuda de IA logré crear un FrameWork escalable, adaptable y ya probado con éxito.

Aprendí no solo de desarrollo embebido y programación sino también de ingeniería inversa, diseño de PCB y control de señales en entornos no amigables.