UNIVERSIDAD DE COSTA RICA

ESCUELA DE INGENIERÍA ELÉCTRICA

IE0624
Laboratorio de Microcontroladores

Laboratorio III

Arduino: PID, GPIO, ADC y comunicaciones

Estudiantes:
Kendall Saborio Picado - B87103
Alexander Rojas Brenes - B86869

Profesor:
Ing. Marco Villalta Fallas

Fecha de entrega: 08/05/2021

Índice

1. INTRODUCCIÓN

Este trabajo aborda el desarrollo de una incubadora de huevos automatizada. Utilizando la plataforma Arduino UNO, se diseña con el objetivo principal de mantener la temperatura óptima para la incubación de huevos dentro del rango [30, 42]℃. Para alcanzar este propósito, se implementó un control PID que regula la temperatura, siendo configurado mediante prueba/error para garantizar un desempeño eficiente. El sistema incluye una pantalla LCD PCD8544 para visualizar la temperatura actual, el valor de control del PID y la temperatura medida. Además, se han incorporado LEDs de alarma para indicar situaciones fuera del rango óptimo de temperatura: azul para temperaturas inferiores a 30℃, rojo para temperaturas superiores a 42℃ y verde para temperaturas dentro del rango deseado. Además se ha integrado un sistema que permite el registro de datos de temperatura en un archivo de texto plano. Este registro se realiza mediante la comunicación serial con un programa Python que captura y almacena los datos en formato CSV, además de proporcionar visualización gráfica de la temperatura de referencia, la salida del PID y la salida del sistema de control. El análisis de las gráficas obtenidas muestra un comportamiento satisfactorio del sistema diseñado. Los resultados validan la efectividad del diseño propuesto para la incubadora automatizada, demostrando su capacidad para mantener condiciones estables de temperatura dentro del rango requerido para la incubación de huevos. Para consultar el trabajo realizado, puede consultarse el siguiente repositorio de trabajo, en la rama llamada “Lab3”: https://github.com/SaboEIE/IE-0624_IS_2024_B87103. El código en C, archivo de simulación .simu, el archivo READ.md y el Makefile puede consultarse en la ruta `Laboratorio_03/src`.

2. NOTA TEÓRICA

Antes de iniciar con el uso y manipulación del lenguaje de programación C para la creación de los algoritmos objetivo del presente laboratorio, es necesario tener a mano una serie de conceptos importantes, los cuales se explican a continuación:

2.1. Arduino UNO

2.2. Microcontrolador ATM328P

Tal como se detalla en la hoja del fabricante [1], este componente corresponde a un microcontrolador de 8 bits, de alto rendimiento y de baja potencia, producido por la empresa Atmel. Cuenta con 131 instrucciones, una memoria flash con capacidad de 32 kb, una SRAM de 2Kb y una memoria EEPROM de 1 Kb. Asimismo, cuenta con 23 GPIO programables, 3 timers (dos de 8 y otro de 16 bits), seis canales PWM, además de SPI y una USART *full duplex*. En la Figura 1 puede observarse la distribución de pines del microcontrolador y sus respectivas funciones:

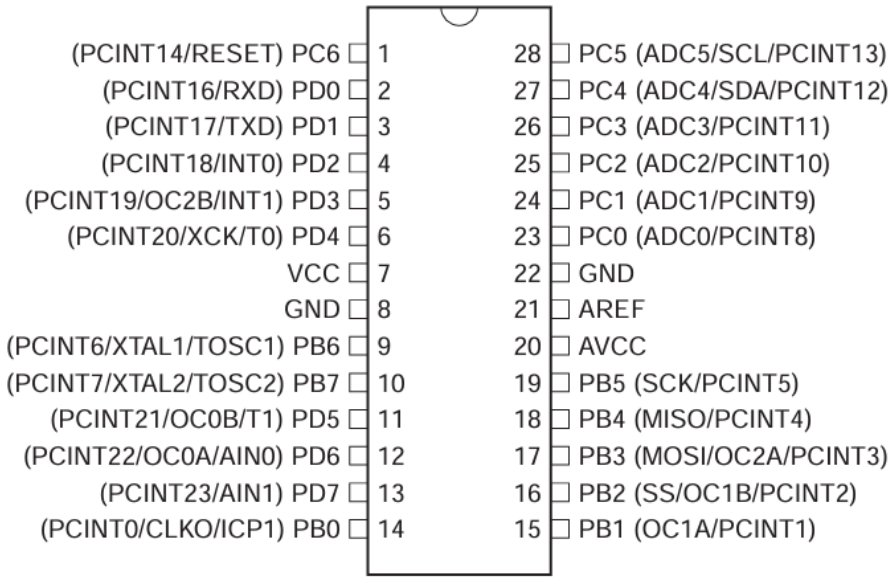


Figura 1: Diagrama de pines y sus respectivas funciones. (Fuente: Imagen tomada de [1])

Para más información general del dispositivo, observar el Anexo 5.1. Por otro lado, en la Figura 2 puede consultarse el diagrama de bloques del microcontrolador en el cual puede consultarse a alto nivel la conexión interna de este (no incluye el CPU):

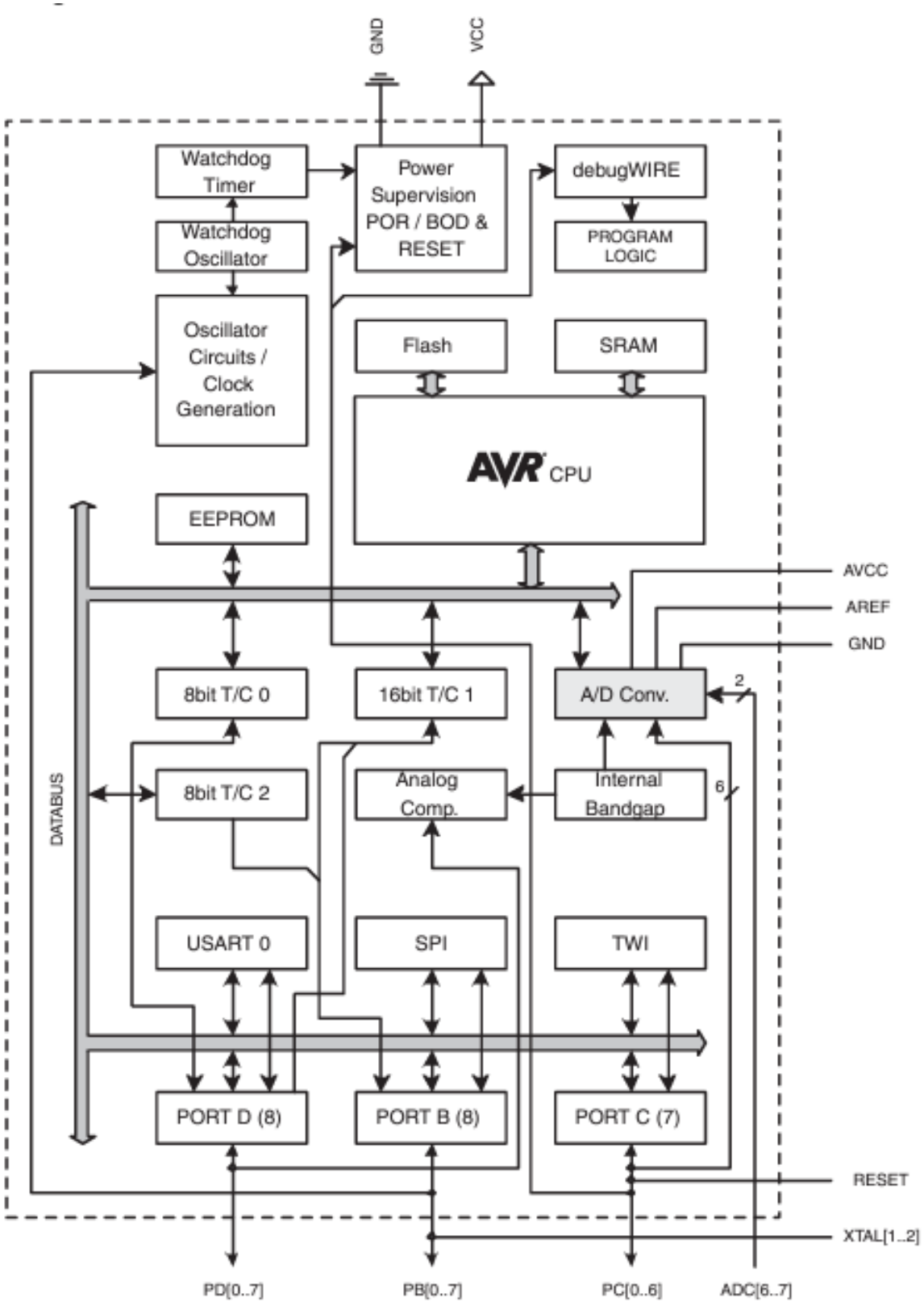


Figura 2: Diagrama de bloques del ATtiny4313. (Fuente: Imagen tomada de [1])

2.2.1. Registros de importancia

A diferencia de otros microcontroladores, Arduino busca hacer la programación y el desarrollo de proyectos de hardware más accesible para una amplia gama de personas. Por lo tanto, se enfoca en proporcionar una capa de abstracción más alta sobre el hardware, lo que significa que no es necesario manipular los registros del microcontrolador directamente en la mayoría de los casos.

2.3. Librerías de importancia

2.3.1. PCD8544

PCD8544 es una biblioteca de software diseñada para Arduino, que permite controlar fácilmente la pantalla Nokia 5110 que cuenta con una resolución de 48x84 píxeles. Esta pantalla es comúnmente utilizada por su bajo costo y bajo consumo, lo que la hace óptima para proyectos y prototipos. La librería proporciona a los desarrolladores una serie de funciones para interactuar con la pantalla, inicializar la pantalla, ajuste de contraste y temperatura, dibujar pixeles, lineas, cuadros, texto de distintos tamaños son algunas de las funcionalidades. Dado que esta destinada para Arduino, su uso es muy intuitivo, esto permite al usuario centrarse en la logica de su aplicación [2].

2.3.2. PID_v1_bc

Esta librería es una versión mejorada de la librería PID, pues se añade una tecnica llamada back calculation que se refiere a un método específico para manejar el windup del integrador en controladores PID. Esta librería ofrece una forma sencilla de integrar este tipo de control PID en proyectos de Arduino, proporcionando una interfaz para configurar los parámetros del controlador PID [3].

2.4. Resistencia de *pull-down*

Del enunciado del laboratorio, se sabe que el objetivo de este es construir un simulador de un paso peatonal que cuente con dos botones de entrada con los que el usuario pueda solicitar una parada del tránsito vehicular. De lo anterior se puede deducir que al menos uno de los pines del microcontrolador debe ser configurado como entrada y que esta debe permanecer en un estado determinado, para que sólo en caso de presionar el botón (se obtiene el inverso del estado anterior), se active su funcionamiento. Para ese diseño, se quiere que el estado por defecto sea en bajo para proteger el circuito constantemente y que el estado en alto (al presionar el botón) active el circuito.

Una resistencia de pull-down permite lograr el comportamiento deseado para el caso expuesto. En la Figura 3 puede observarse la disposición adecuada construida en SimulIDE según se detalla en [4]:

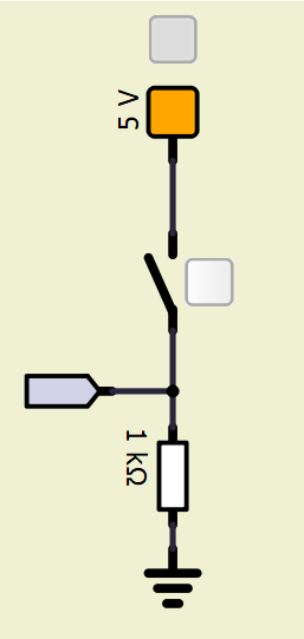


Figura 3: Ejemplo de resistencia de *pull-down* y *switch* en SimulIDE.

En la Figura 3, puede observarse como el pulsador está conectado entre la fuente de tensión y el pin del MCU. Cuando el interruptor es presionado, la entrada del microcontrolador está en un valor lógico alto, pero cuando el interruptor está abierto, la resistencia *pull-down* "jala" la tensión de entrada a la tierra (valor lógico cero). Según se explica en [4], la resistencia *pull-down* debe ser mayor que la impedancia de salida del dispositivo, ya que de lo contrario podría ser capaz de bajar la tensión demasiado y la tensión de entrada en el pin se mantendría en un valor bajo lógico constante. Para calcular esta, se aplica la ley de Ohm de la siguiente forma:

$$Z_{MCU} = \frac{V_{DD}}{I_{max}} \tag{1}$$

Donde V_{DD} corresponde a la tensión máxima entregada por cada pin del MCU y I_{max} la corriente máxima en el mismo caso, de esta forma se tiene que:

$$Z_{MCU} = \frac{5}{20 \times 10^{-3}} = 250\ \Omega \tag{2}$$

Para este caso particular, se tomará su valor como 1 kΩ para asegurarse de que no haya ningún problema.

2.5. Circuito para lidiar con el *switch bounce*

Tal como se detalla en [5], cerrar un *switch*, puede parecer que este hace contacto de manera absoluta e inmediata, sin embargo, este dispositivo no deja de ser mecánico y el contacto ocurre de manera oscilante, por lo que una serie de interferencias son ingresadas a la señal transmitida. Para solucionar este problema, puede utilizarse un circuito RC a la salida del interruptor, de tal forma que se filtren los picos producidos por las oscilaciones y suavicen la señal que será transmitida. Lo

anterior es útil para evitar falsas pulsaciones. Un ejemplo del tipo de circuito a construir es el que puede observarse en la Figura 4:

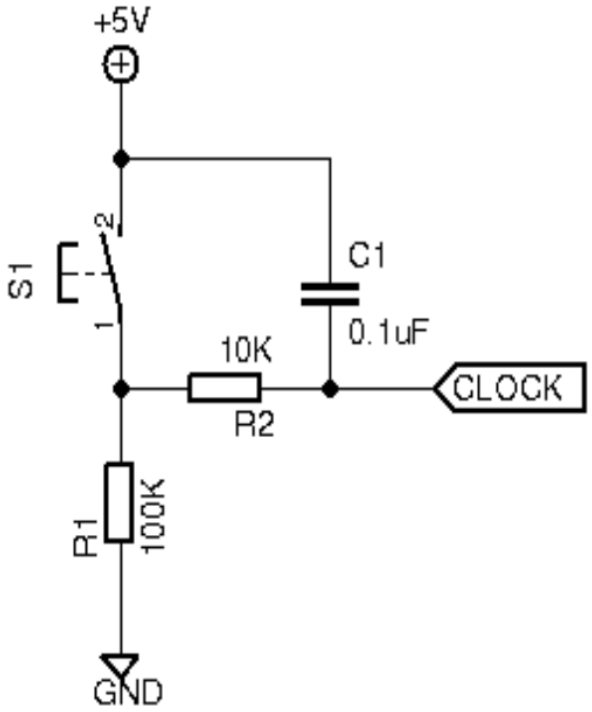


Figura 4: Ejemplo de circuito utilizado para evitar las oscilaciones producidas por un *switch* que combina un circuito RC con un resistor de *pull-down*. (Fuente: Imagen tomada de [5])

Para hacer los cálculos adecuados de capacitancia y resistencia, se inicia definiendo un tiempo de carga y descarga adecuado. En este caso, como no se trata de una acción que necesite de actividad inmediata, se puede trabajar en el orden de los milisegundos, en este caso, de los 100 ms. Como bien se sabe:

$$\tau = R \cdot C \tag{3}$$

Por lo que si ya se tiene un valor de tiempo, basta con seleccionar una de las dos variables restantes para despejar la otra. Si se toma un valor de resistencia de 5 kΩ, entonces despejando:

$$C = \frac{\tau}{R} = \frac{10 \times 10^{-3}}{5 \times 10^{3}} = 2\ \mu F \tag{4}$$

Para apegarse a los valores disponibles en el mercado, se elige un capacitor de 2.2 μF.

El circuito resultante construido en SimulIDE puede consultarse en la Figura 5 a continuación:

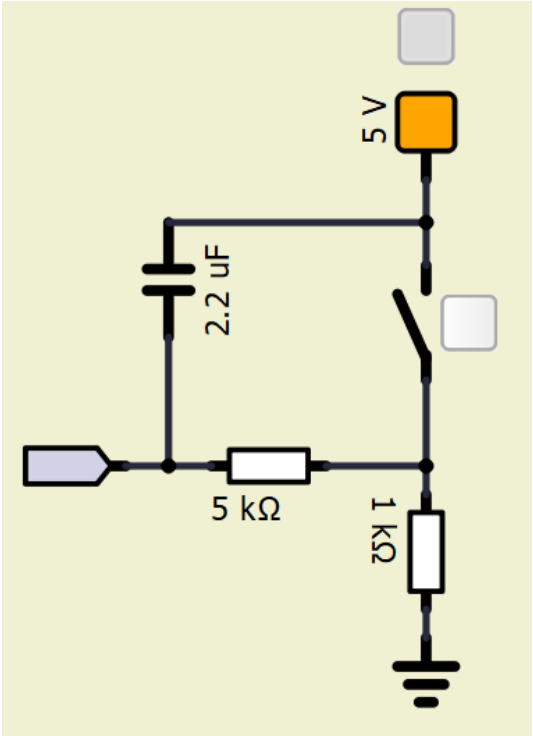


Figura 5: Circuito utilizado para evitar las oscilaciones producidas por un *switch* que combina un circuito RC con un resistor de *pull-down*.

2.6. Diseño del circuito simulador de una incubadora

Según se indica en el enunciado, el presente laboratorio tiene como objetivo simular el funcionamiento de un una incubadora de huevos automática basada en el Arduino UNO.

Figura 6: Incubadora Automática.

Algunas de las características solicitadas en el diseño importantes para la construcción del circuito son poder visualizar la temperatura en una pantalla LCD PCD8544, utilizar un potenciómetro para establecer la temperatura de operación, encender un LED de alarma color azul si la temperatura de la incubadora (T) es menor a 30 ℃, encender un LED de alarma color rojo si T $>$ 42 ℃, encender un LED color verde si 30 ℃ $\leq T \leq$ 42 ℃, un *switch* para habilitar la comunicación con la PC y otro para habilitar la impresión en la pantalla LCD.

De la características solicitadas se sabe que habrán 3 LEDs conectados entre alguno de los pines del Arduino y la tierra, por lo que estos necesitan de resistencias de protección. Para calcular el valor de estas, se debe tomar en cuenta que la tensión máxima de operación es de 5 V y la corriente máxima suministrada es de 20 mA, información tomada de la hoja del fabricante [1]. Para obtener el valor de la resistencia de protección utilizando la ley de Kirchhoff de esta forma:

$$R = \frac{V_{DD} - V_{LED}}{I_{max}} \tag{5}$$

Como se trata de una simulación, se tomará V_{LED} = 2.4 V (tensión de umbral del simulador) para todos los casos. Para no presionar el sistema, se realizarán los cálculos utilizando 18 mA. De esta forma:

$$R = \frac{5 - 2{,}4}{18 \times 10^{-3}} = 144{,}44\ \Omega \tag{6}$$

Es este caso, se selecciona un valor estándar de 150 Ω. Para lo que sería la conexión de la pantalla LCD Nokia 5110 a arduino, simplemente se debe conectar según se establece en la documentación de la librería PCD8544.[2] A continuación una tabla con los respectivos pines de Arduino y la pantalla:

Tabla 1: Conexión de pines pantalla-arduino

Pines LCD NOKIA 5110	Pines Arduino UNO
RST	7
CE	6
DC	5
Din	4
CLK	3

2.7. Lista de componentes y precios

En la Tabla 2 pueden consultarse los componentes utilizados y sus precios, tomando como referencia los precios del sitio web de la tienda de componentes MicroJPM (https://www.microjpm.com/):

Tabla 2: Lista de componentes.

Componente	Cantidad	Precio
Placa Arduino UNO	1	$19,90
Switches	2	$0,30
Capacitor de 2.2 μF	2	$0,35
Resistor de 5 kΩ	2	$0,07
Resistor de 1 kΩ	2	$0,07
Resistores de 150 Ω	3	$0,13
LED azúl	1	$0.14
LED verde	1	$0.51
LED rojo	1	$0,51
Pantalla LCD	1	$7,50
Potenciómetro 1K	1	$1,00
Total	-	$31.53

Según la Tabla 2, para realizar este proyecto son necesario ₡16,122.71, según el valor del dólar al momento de escribir el presente informe.

3. DESARROLLO Y ANÁLISIS DE RESULTADOS

A continuación se muestran los métodos de desarrollo utilizados para resolver cada uno de los problemas planteados, así como la funciones utilizadas, el método de operación a partir de diagramas de flujo y los resultados obtenidos:

3.1. Desarrollo de la solución

A continuación podrá consultarse una descripción de alto nivel de cada una de las funciones implementadas para conseguir los objetivos del presente laboratorio:

3.1.1. Función `setup()`

Esta función se encarga de inicializar todo lo necesario para que el proyecto funcione correctamente.Configura los pines de salida para los LEDs y de entrada para los dispositivos LCD y Serial, además inicia la comunicación con el LCD y configura el modo de operación del controlador PID en modo automático.

3.1.2. Función `simPlanta()`

Esta función se proporcionada por el enunciado. Simula el comportamiento térmico de un bloque de aluminio con calentamiento y enfriamiento pasivo, que vendría a tomar el comportamiento de la incubadora. Dentro de la función se calcula la temperatura del bloque de aluminio en función del tiempo, considerando la transferencia de calor desde un calentador y el enfriamiento por convección al ambiente. La función recibe como parámetro la cantidad de calor `Q` suministrado al bloque de aluminio (J/s). Esta función retorna la temperatura actual `T` del bloque de aluminio (℃).

3.1.3. Función `LED_control()`

Esta función controla el encendido y apagado de los LEDs en base a la temperatura externa medida en la planta. La función recibe como parámetro `out_temp` Temperatura de salida medida (en grados Celsius).

3.1.4. Función `LCD_control()`

Esta función actualiza la pantalla LCD con las temperaturas operativa, de salida y la salida del controlador PID (Señal de control). La función recibe como parámetros `op_temp` temperatura operativa actual (en grados Celsius), `PID` salida del controlador PID, `out_temp` temperatura de salida actual de la planta (en grados Celsius)..

3.1.5. Función `serial_com()`

Función que imprime valores en el puerto serial. Esta función agrupa los valores de la temperatura de operación, la señal de control y la señal de salida de la planta, de forma que estos puedan ser captados y utilizados en futuro procesamiento. Cuenta con tres parámetros de tipo `double`:

`op_temp`, `PID` y `out_temp`, los cuales representan la temperatura de referencia, la señal de control y la temperatura de salida de a planta.

3.1.6. Función `loop()`

`loop()` es la función principal del programa, realiza las siguientes operaciones en un ciclo continuo:

- Lee el valor del potenciómetro.
- Mapea el valor leído a un rango de 20 a 80.
- Actualiza el valor de referencia del controlador PID (Setpoint).
- Calcula la salida del controlador PID.
- Convierte la salida en temperatura del controlador PID a watts.
- Simula el comportamiento térmico de la planta y obtiene la temperatura de salida.
- Controla los LEDs en función de la temperatura de salida.
- Actualiza la pantalla LCD si el pin de control está en estado alto.
- Inicia o detiene la comunicación serial si el pin de control está en estado alto.

3.1.7. Diagrama de Flujo

A continuación se muestra un diagrama de flujo del código en C utilizado para este laboratorio:

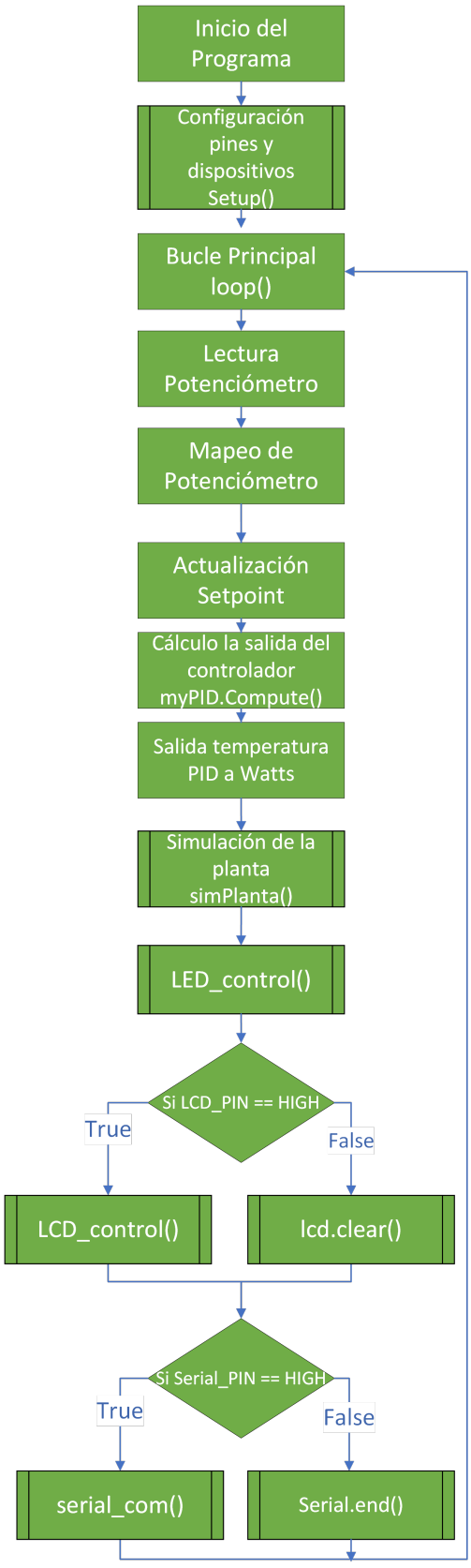


Figura 7: Diagrama de flujo que describe el funcionamiento del sistema.

3.1.8. *Script* de Python `datasaver.py`

Este script de Python se encarga de leer datos desde un puerto serial, el cual está conectado al Arduino UNO utilizado para realizar el proyecto, y almacenar esos datos en un archivo CSV para su posterior análisis. Primero, establece la comunicación con el puerto serial especificado y abre un archivo CSV en modo escritura. Luego, registra el tiempo inicial y entra en un bucle infinito donde lee continuamente datos del puerto serial. Cada dato es procesado para eliminar información innecesaria y se agrega el tiempo transcurrido como el primer campo. Estos datos procesados se imprimen en la terminal para verificación y se escriben en el archivo CSV. El *script* opera de manera continua, asegurando una recopilación constante de datos.

3.1.9. *Script* de Python `graph.py`

Este script de Python utiliza la biblioteca `csv` para cargar datos desde un archivo CSV y también `matplotlib.pyplot` para visualizar esos datos en forma de gráficos. Después de cargar los datos del archivo CSV en listas separadas para cada columna, utiliza `matplotlib` para graficar cada señal respecto al tiempo (primera columna). Finalmente, el *script* muestra el gráfico generado.

3.1.10. *Script* de Bash `virt_port.sh`

Este *script* utiliza el comando `socat` para crear un puerto virtual (PTY) que enlaza dos dispositivos virtuales, `/tmp/ttyS0` y `/tmp/ttyS1`. Estos puertos virtuales se configuran para comunicarse en modo “raw”, lo que significa que los datos se transmitirán sin procesamiento adicional. Además, se deshabilita la retroalimentación de eco (`echo=0`), lo que evita que los caracteres enviados se muestren nuevamente en el terminal. Lo anterior permite la extracción desde el Arduino.

3.1.11. Construcción del circuito utilizado

Para iniciar con la construcción de este circuito, es necesario tomar en cuenta los métodos de protección descritos en las secciones 2.4 y 2.5. Una vez que ambos fueron unidos y construidos, se conectó la salida de este circuito al pin A5 (se utiliza como pin digital), de forma que este funcionara como un interruptor para habilitar y deshabilitar la comunicación serial. Otra instancia del circuito mencionado fue conectada al pin A4 (se utiliza como pin digital) y en este caso, se utiliza para apagar y encender la pantalla LCD utilizada para mostrar las temperaturas y datos de control. El método de cálculo de las resistencias de cada LED puede observarse en la sección 2.6. Por otro lado, en el caso de los LEDs utilizados para notificar del estado de la señal de salida, se utilizaron los pines D0-D2, los cuales son pines digitales y son óptimos para esta aplicación. Asimismo, en el pin A0 (utilizado como un ADC), se conectó un potenciómetro utilizado para definir la temperatura de referencia, el valor es indiferente, ya que internamente se trabaja con un mateo analógico. Finalmente, la disposición de pines para la LCD utilizada en el laboratorio también puede consultarse en la sección 2.6. El circuito resultante tras implementar todas as consideraciones anteriores puede consultarse en la Figura 8:

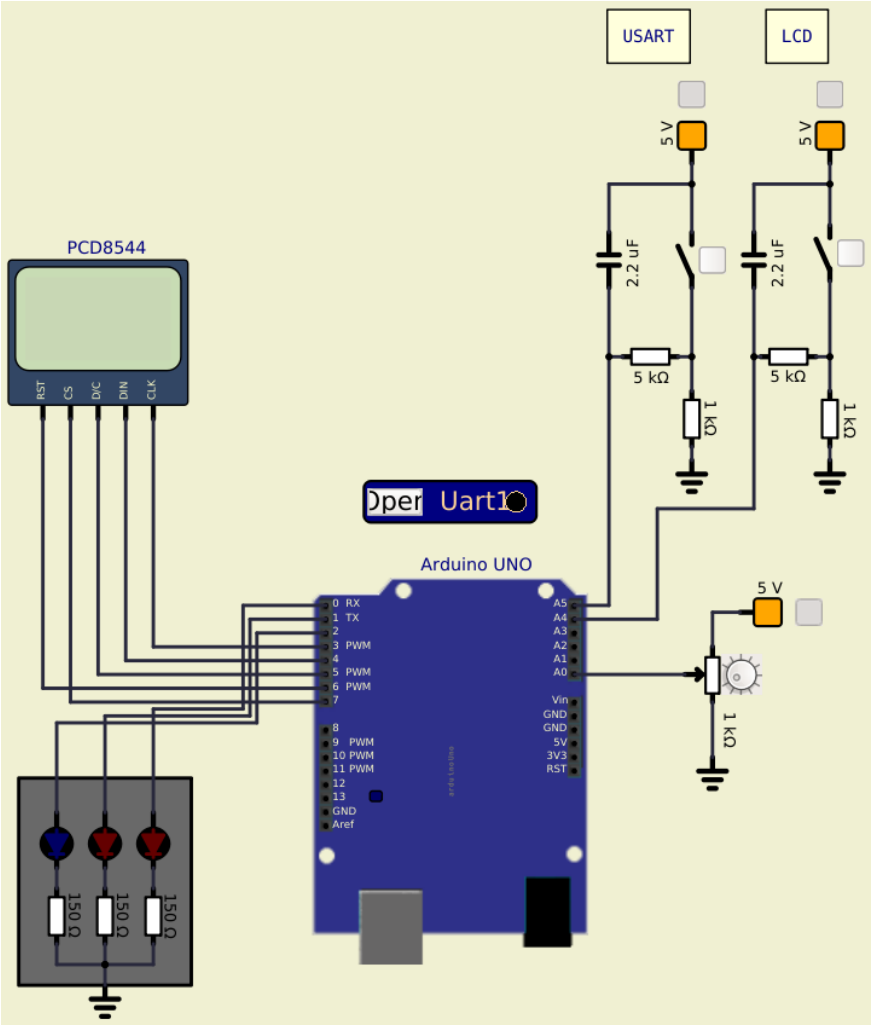


Figura 8: Circuito final tras los cálculos de magnitudes y la implementación de las consideraciones necesarias.

3.2. Análisis de los resultados

A continuación se mostrarán los resultados para cada una de las funcionalidades solicitadas por el enunciado:

3.2.1. Temperatura bajo el rango adecuado

En la Figura 9 puede observarse el comportamiento de los LEDs en función de la temperatura de salida de la planta cuando esta está por debajo del rango óptimo ($30\,℃ \leq T \leq 42\,℃$):

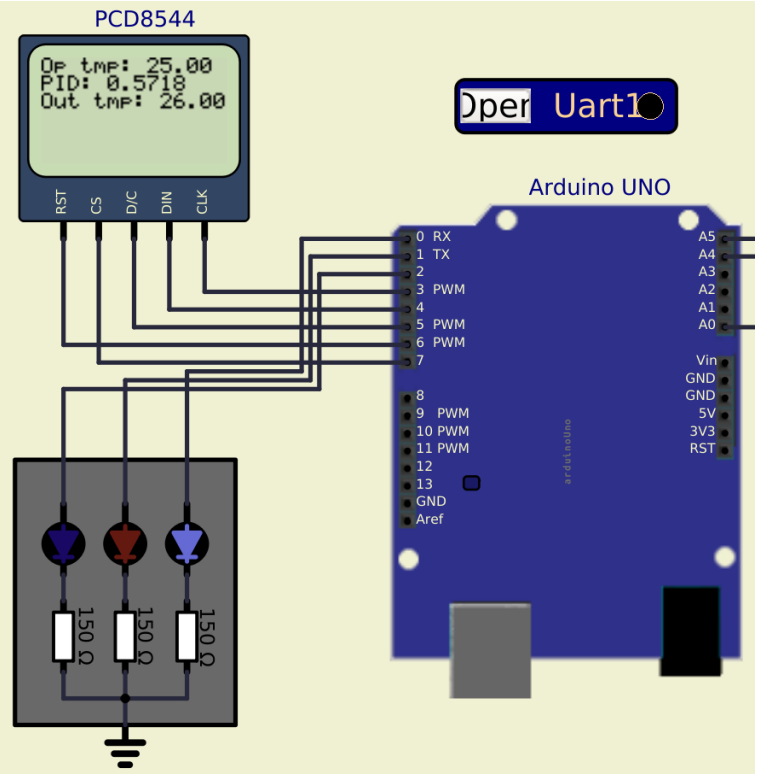


Figura 9: Funcionamiento del sistema cuando la temperatura de salida de la planta es inferior al rango óptimo de temperatura.

Como puede verse en la Figura 9, el LED azul efectivamente se enciende cuando la temperatura de salida de la planta se encuentra en por debajo del rango óptimo.

3.2.2. Temperatura dentro del rango adecuado

En la Figura 10 puede observarse el comportamiento de los LEDs en función de la temperatura de salida de la planta cuando esta está dentro del rango óptimo (30 ℃ $\leq T \leq$ 42 ℃):

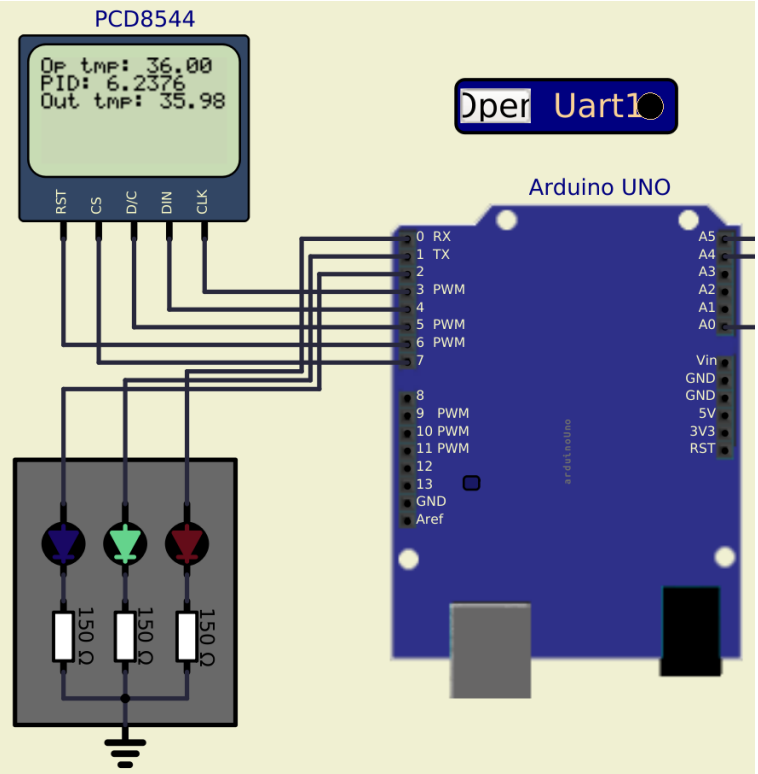


Figura 10: Funcionamiento del sistema cuando la temperatura de salida de la planta se encuentra dentro al rango óptimo de temperatura.

Como puede verse en la Figura 10, el LED verde efectivamente se enciende cuando la temperatura de salida de la planta se encuentra dentro del rango óptimo.

3.2.3. Temperatura sobre el rango adecuado

En la Figura 10 puede observarse el comportamiento de los LEDs en función de la temperatura de salida de la planta cuando esta está por encima del rango óptimo (30 °C $\leq T \leq$ 42 °C):

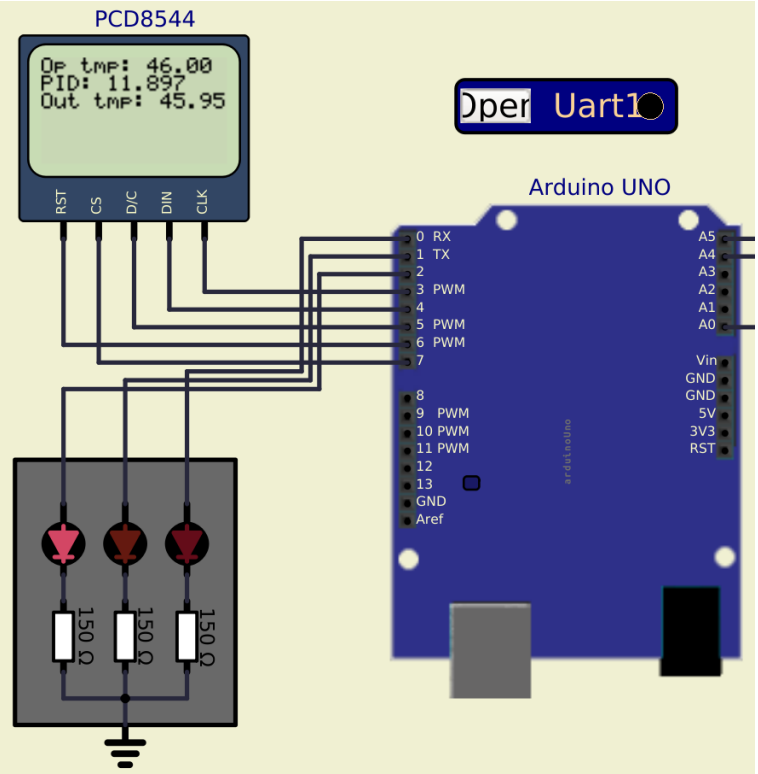


Figura 11: Funcionamiento del sistema cuando la temperatura de salida de la planta es superior al rango óptimo de temperatura.

Como puede verse en la Figura 11, el LED rojo efectivamente se enciende cuando la temperatura de salida de la planta se encuentra en por encima del rango óptimo.

3.2.4. Desactivación de la pantalla

En la Figura 12 puede observarse el comportamiento del sistema cuando el *switch* que activa la pantalla no fue activado:

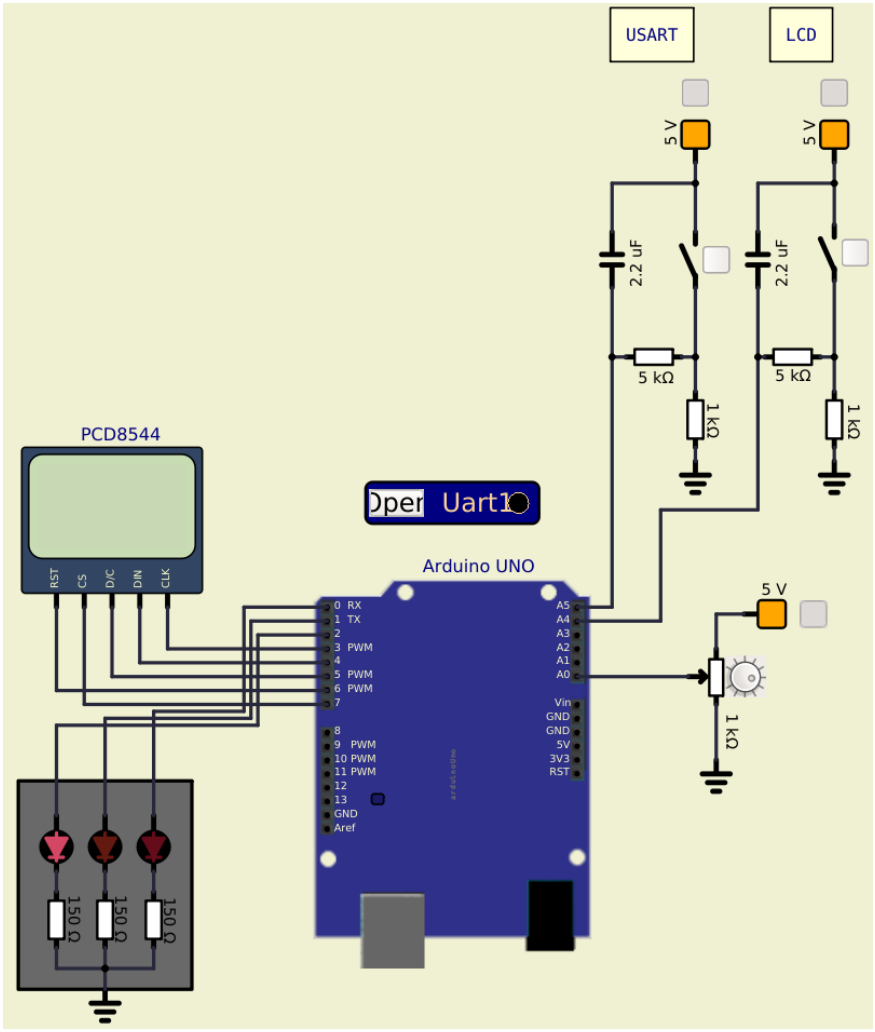


Figura 12: Funcionamiento de la pantalla LCD cuando el *switch* encargado de gestionarla no fue activado.

Como puede verse en la Figura 12, si el *switch* encargado de gestionar el funcionamiento de la pantalla no ha sido activado, esta no proyecta ninguna información y se mantiene apagada, a pesar de que puede verse que el circuito sigue en funcionamiento, ya que el LED rojo está encendido, indicando que se está trabajando en una temperatura superior al rango óptimo.

3.2.5. Desactivación de la comunicación serial

A continuación puede verse el estado de la comunicación serial cuando el *switch* que activa la comunicación serial no fue activado (Figura 13) y el caso contrario (Figura 14):

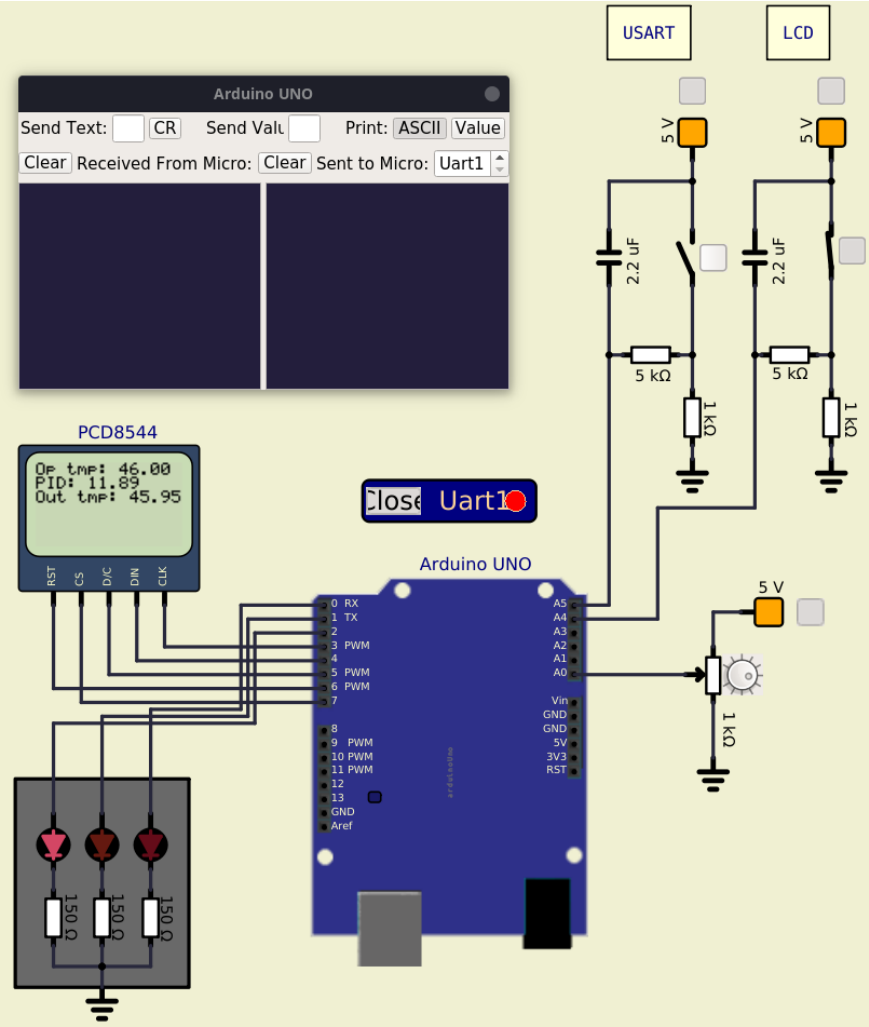


Figura 13: Funcionamiento de la salida de datos por el puerto serial cuando el *switch* encargado de gestionarlo no fue activado.

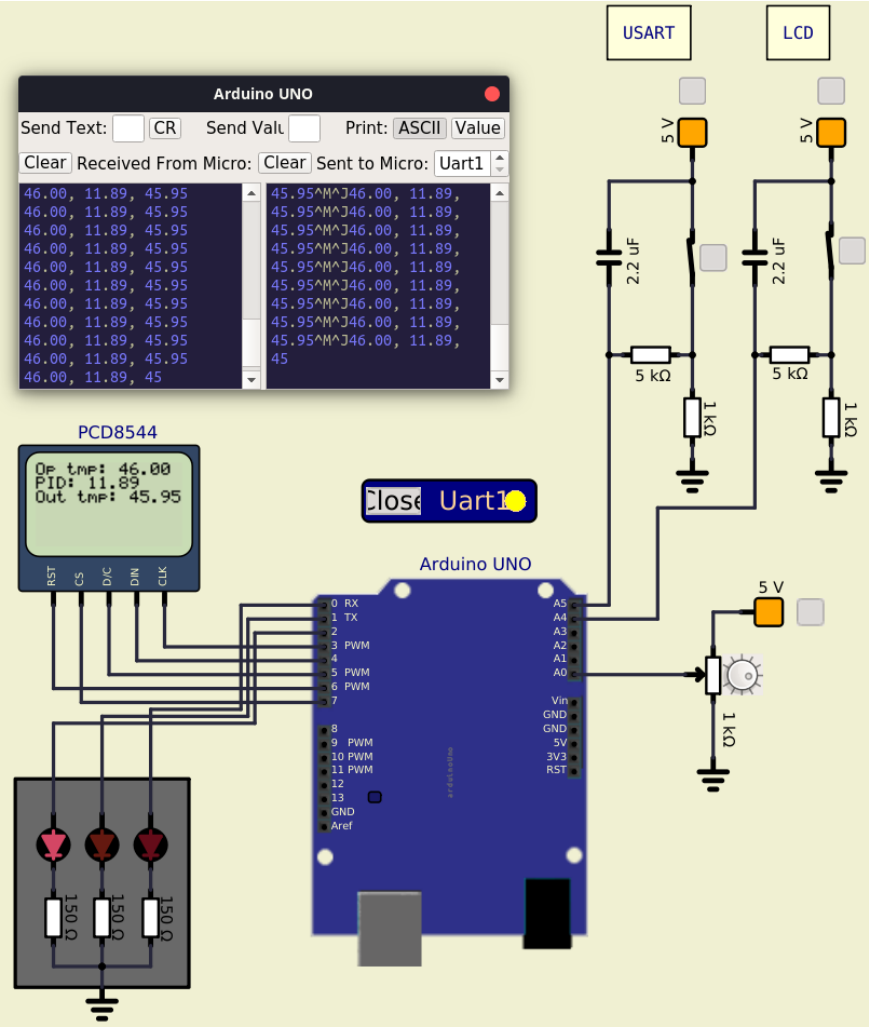


Figura 14: Funcionamiento de la salida de datos por el puerto serial cuando el *switch* encargado de gestionarlo fue activado.

Como puede verse en las Figuras 13 y 14, si el *switch* encargado de gestionar el funcionamiento de la comunicación serial ha sido activado, los datos recopilados por el Arduino son transportados a la computadora a través del puerto virtual y en caso contrario, esto no ocurre.

3.2.6. Aumento de temperatura

A continuación pueden observarse el punto de inicio y final (Figuras 15 y 16) y el comportamiento de las señales de temperatura de referencia, señal de control y temperatura de salida antes una disminución en la temperatura (Figura 17):

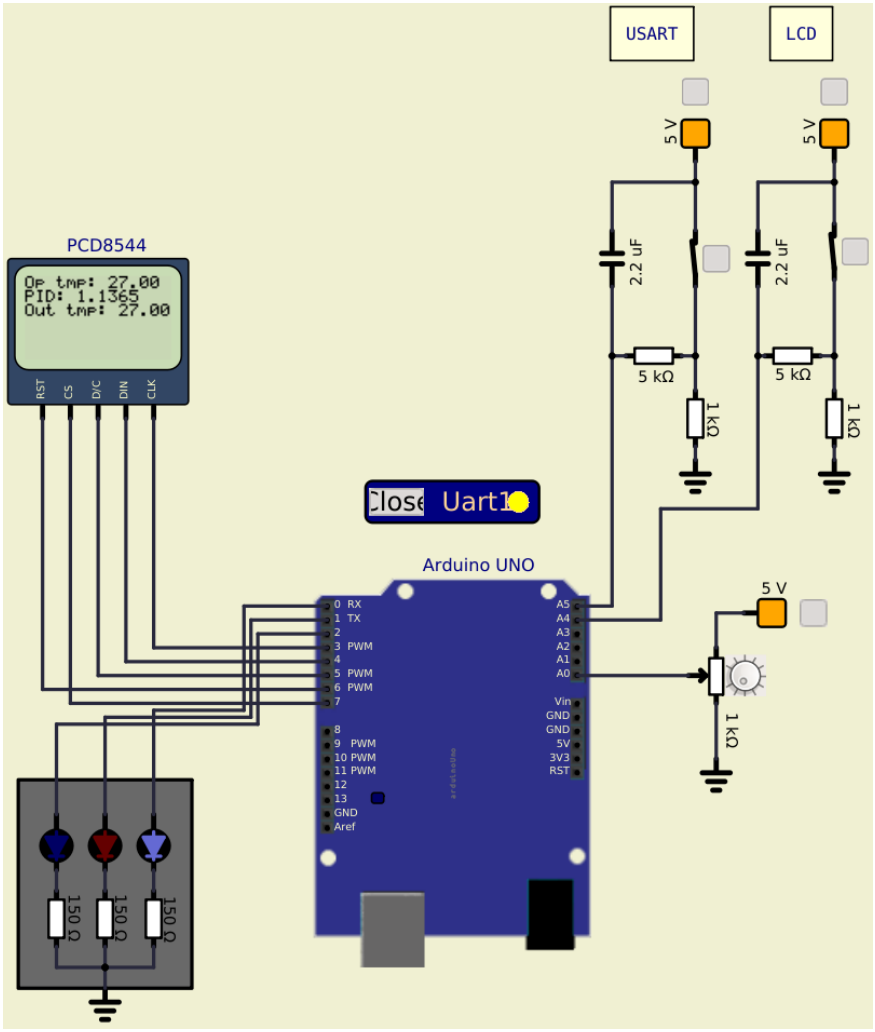


Figura 15: Temperatura de inicio antes del aumento.

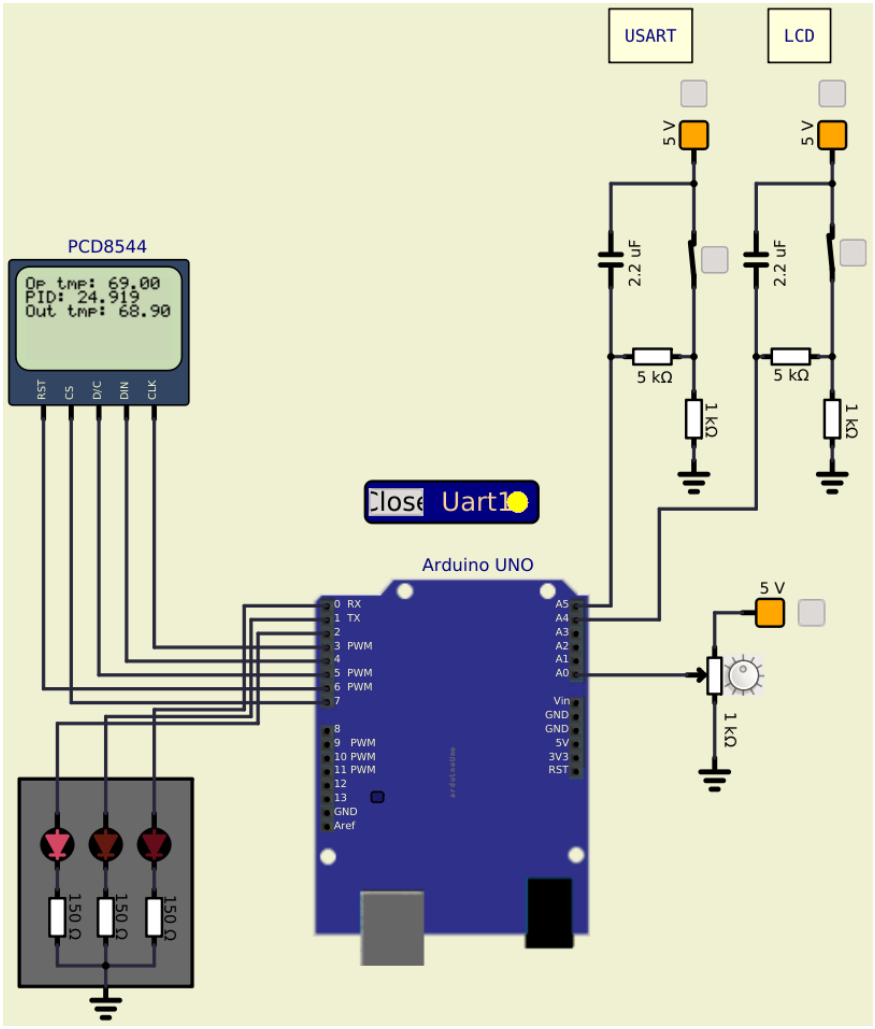


Figura 16: Temperatura de final tras el aumento.

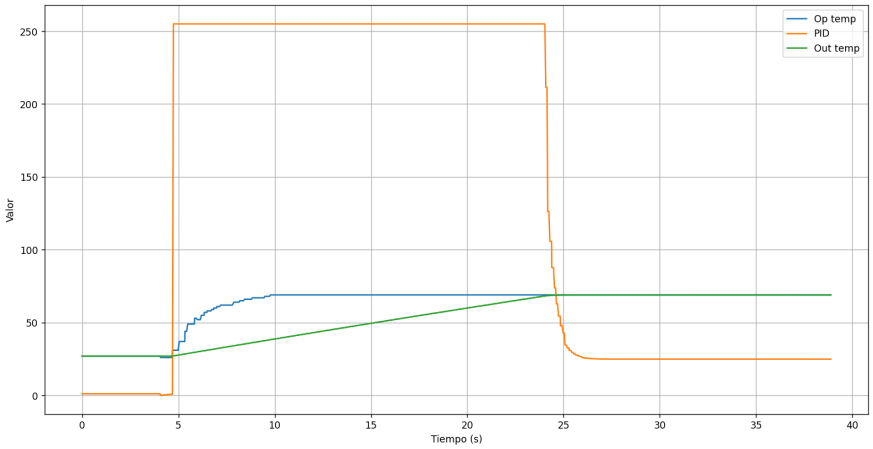


Figura 17: Gráfica de aumento de temperatura para cada una de las señales de interés.

Como puede verse en la gráfica de la Figura 17, la temperatura de salida sigue rápidamente a

la temperatura de referencia, lo cual es un comportamiento esperado (la temperatura debería subir relativamente rápido). Asimismo, se puede ver como la señal de control sube considerablemente para compensar el cambio en la referencia y baja en el momento en el que la señal de salida alcanza el *setpoint*. En la Figura 16 puede verse como la temperatura de salida se aproxima casi de manera absoluta a la referencia.

3.2.7. Disminución de temperatura

A continuación pueden observarse el punto de inicio y final (Figuras 18 y 19) y el comportamiento de las señales de temperatura de referencia, señal de control y temperatura de salida antes una disminución en la temperatura (Figura 20):

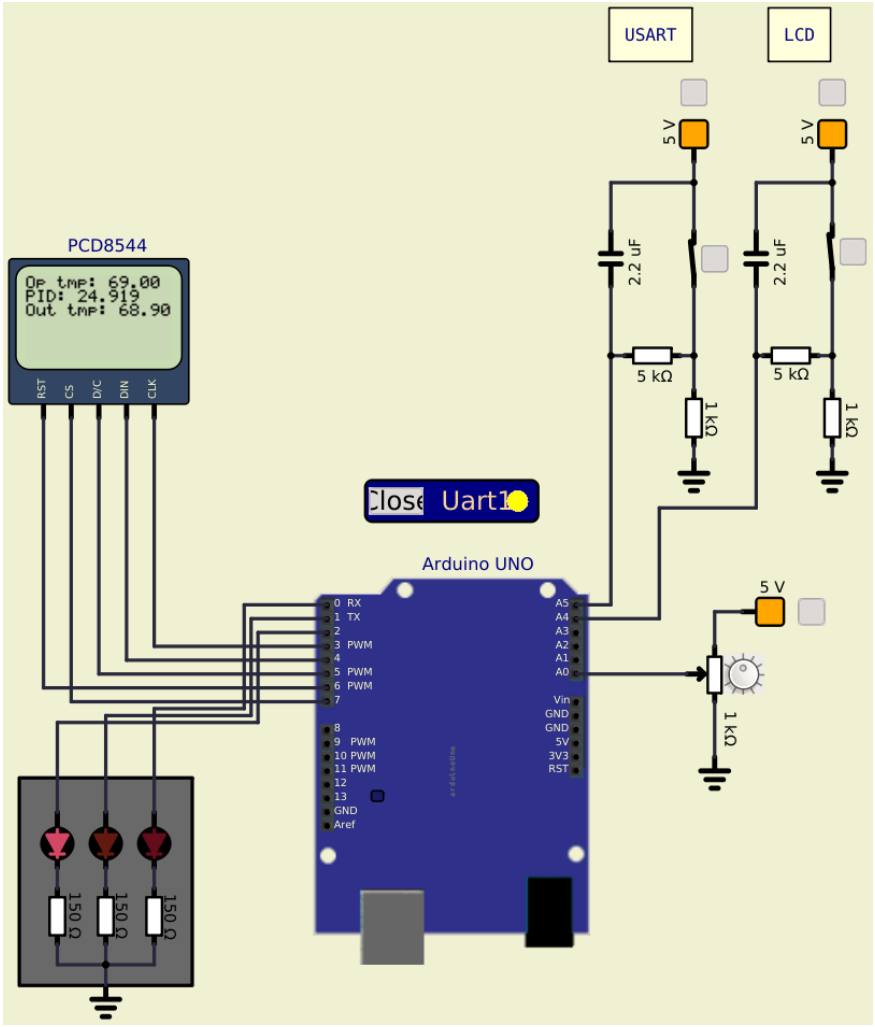


Figura 18: Temperatura de inicio antes de la disminución.

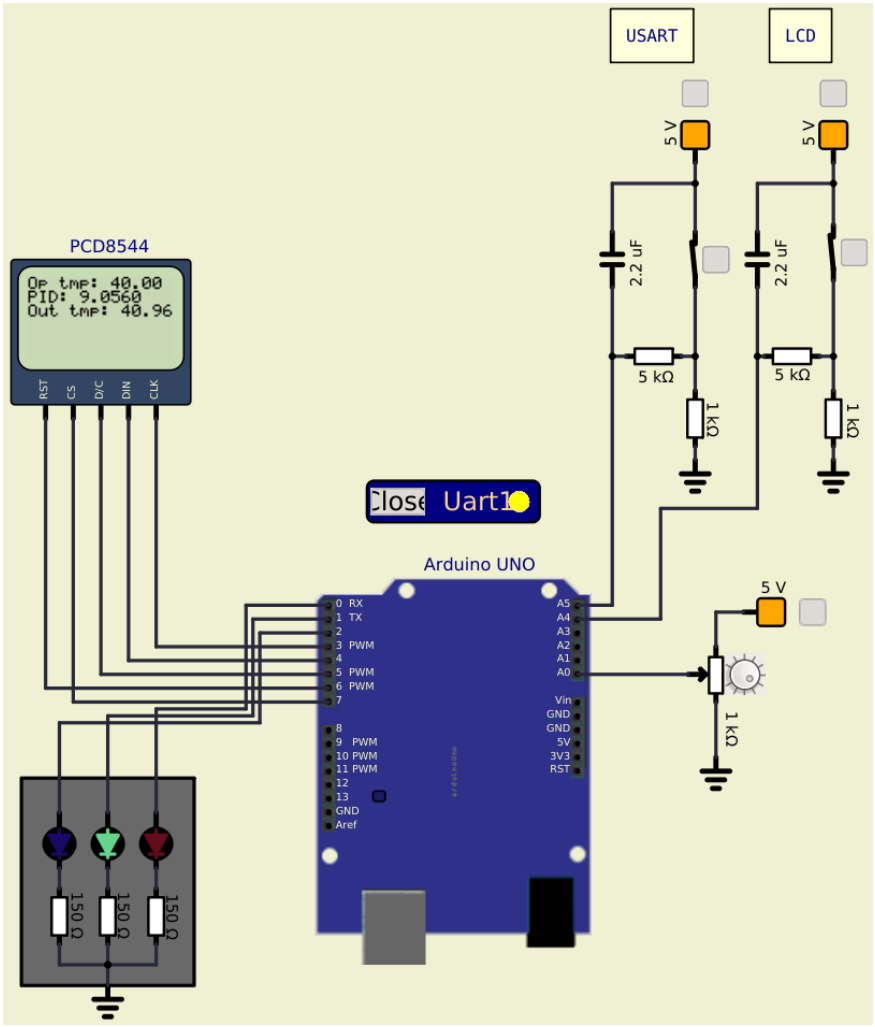


Figura 19: Temperatura de final tras la disminución.

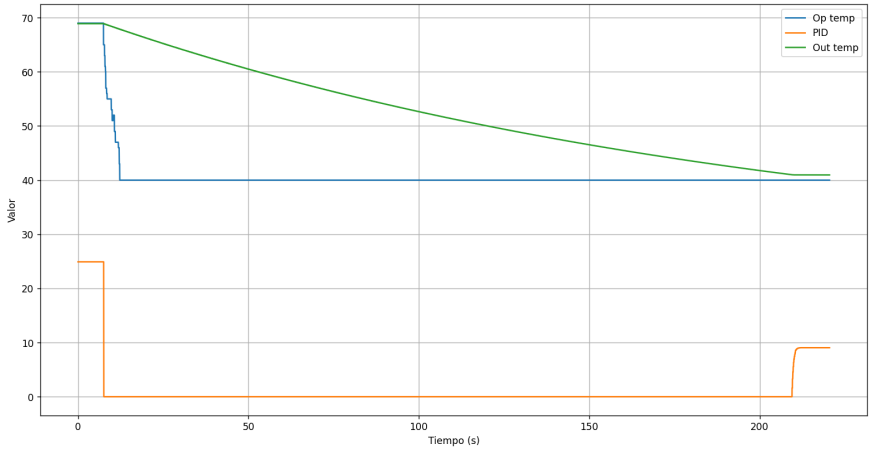


Figura 20: Gráfica de disminución de temperatura para cada una de las señales de interés.

Como puede verse en la gráfica de la Figura 17, la temperatura de salida sigue lentamente a

la temperatura de referencia, lo cual es un comportamiento esperado (la temperatura debería bajar relativamente lento). Asimismo, se puede ver como la señal de control baja para compensar el cambio en la referencia y sube en el momento en el que la señal de salida alcanza el *setpoint*. En la Figura 19 puede verse como la temperatura de salida se aproxima casi de manera absoluta a la referencia.

4. CONCLUSIONES Y RECOMENDACIONES

A lo largo del desarrollo del informe se pusieron a prueba una serie de conceptos y prácticas importantes para el manejo básico del Arduino UNO, así como sus GPIO, convertidores analógico-digitales, comunicación serial y formas de programación. A continuación algunas conclusiones y recomendaciones recolectadas a lo largo de trabajo realizado:

- Se confirma que el sistema responde correctamente a diferentes rangos de temperatura, como se evidencia en la activación de los LEDs correspondientes dependiendo de si la temperatura está por debajo, dentro o por encima del rango óptimo. Esto indica una adecuada detección y respuesta del sistema a las condiciones térmicas requeridas.

- Se observa que la desactivación de la pantalla y la comunicación serial ocurren como se espera cuando los respectivos *switches* no son activados. Esto sugiere un buen funcionamiento de las funciones adicionales de control y comunicación, lo cual es esencial para el funcionamiento global del sistema.

- Los resultados de aumento y disminución de temperatura muestran una respuesta coherente del sistema, donde la señal de control se ajusta apropiadamente para mantener la temperatura de salida cerca de la temperatura de referencia. Esto indica una efectiva regulación térmica y una capacidad de respuesta dinámica del sistema.

- Es recomendable realizar pruebas adicionales en condiciones extremas o inesperadas para evaluar la robustez del sistema. Esto ayudaría a identificar posibles vulnerabilidades o escenarios no contemplados durante el diseño inicial.

- Dado el buen funcionamiento general del sistema, sería beneficioso agregar alarmas o notificaciones para alertar a los usuarios en caso de que la temperatura se salga del rango óptimo durante un tiempo prudencial. Esto es especialmente crítico en el caso de una incubadora, donde desviaciones de temperatura podrían dañar los huevos en desarrollo.

REFERENCIAS BIBLIOGRÁFICAS

[1] Atmel, *ATmega328P: 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash*. 7810D-AVR-01/15, 2015. Disponible en: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf.

[2] C. Rodrigues, "PCD8544." https://www.arduino.cc/reference/en/libraries/pcd8544/.

[3] D. Forrest, "PID_v1_bc." https://www.arduino.cc/reference/en/libraries/pid_v1_bc/.

[4] EEPower, "Pull-up and Pull-down Resistors." https://eepower.com/resistor-guide/resistor-applications/pull-up-resistor-pull-down-resistor/#, s.f. Accessed: April 01, 2024.

[5] All About Circuits, "Switch Bounce and How to Deal with It." https://www.allaboutcircuits.com/technical-articles/%20switch-bounce-how-to-deal-with-it/, 2015. Accessed: April 01, 2024.

5. Anexos

5.1. Hoja del fabricante del ATmega328P Atmel: información general

Atmel

ATmega328P

8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash

DATASHEET

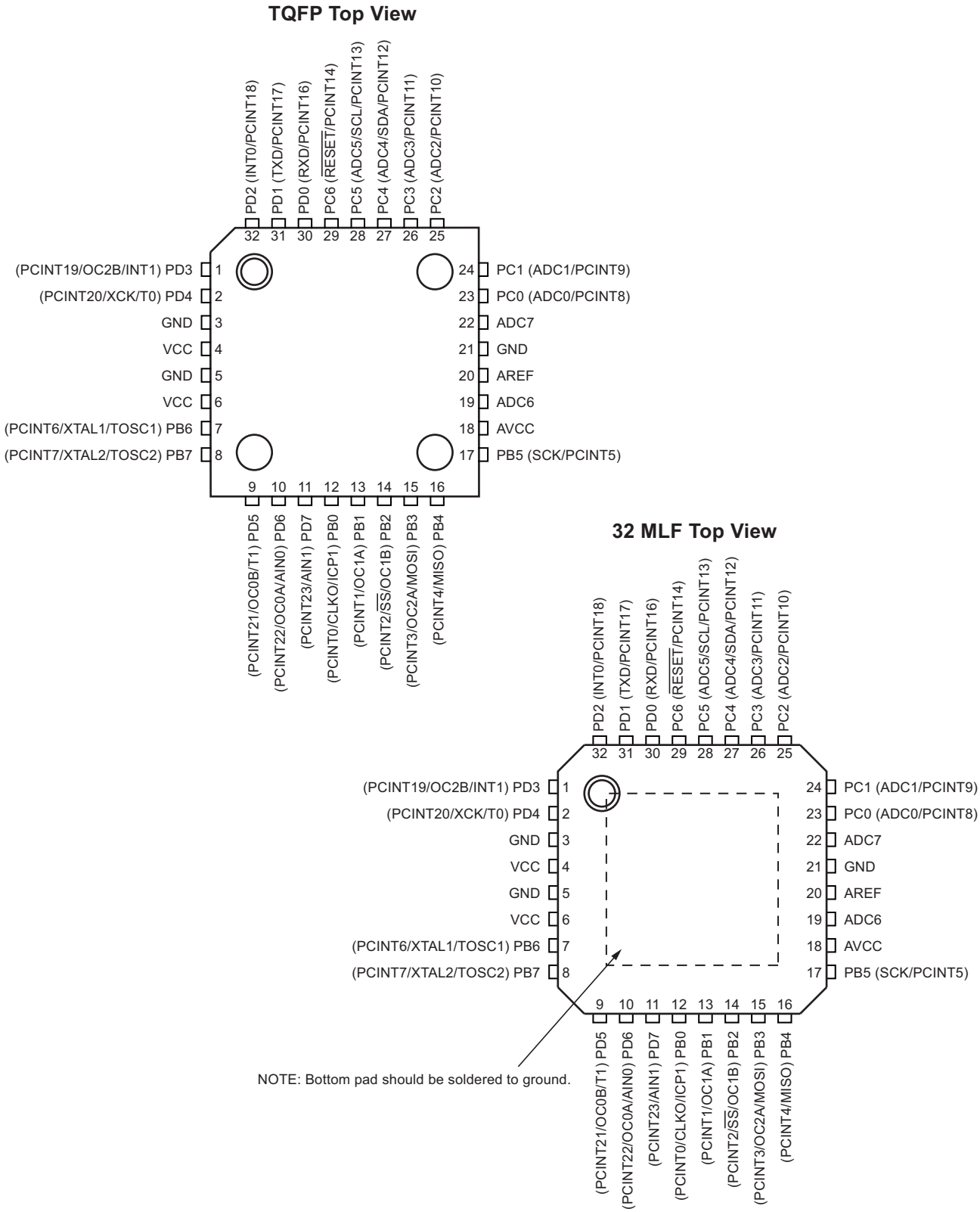
Features

- High performance, low power AVR® 8-bit microcontroller
- Advanced RISC architecture
  - 131 powerful instructions – most single clock cycle execution
  - 32 × 8 general purpose working registers
  - Fully static operation
  - Up to 16MIPS throughput at 16MHz
  - On-chip 2-cycle multiplier
- High endurance non-volatile memory segments
  - 32K bytes of in-system self-programmable flash program memory
  - 1Kbytes EEPROM
  - 2Kbytes internal SRAM
  - Write/erase cycles: 10,000 flash/100,000 EEPROM
  - Optional boot code section with independent lock bits
    - In-system programming by on-chip boot program
    - True read-while-write operation
  - Programming lock for software security
- Peripheral features
  - Two 8-bit Timer/Counters with separate prescaler and compare mode
  - One 16-bit Timer/Counter with separate prescaler, compare mode, and capture mode
  - Real time counter with separate oscillator
  - Six PWM channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    - Temperature measurement
  - Programmable serial USART
  - Master/slave SPI serial interface
  - Byte-oriented 2-wire serial interface (Phillips I^2C compatible)
  - Programmable watchdog timer with separate on-chip oscillator
  - On-chip analog comparator
  - Interrupt and wake-up on pin change
- Special microcontroller features
  - Power-on reset and programmable brown-out detection
  - Internal calibrated oscillator
  - External and internal interrupt sources
  - Six sleep modes: Idle, ADC noise reduction, power-save, power-down, standby, and extended standby

- I/O and packages
  - 23 programmable I/O lines
  - 32-lead TQFP, and 32-pad QFN/MLF
- Operating voltage:
  - 2.7V to 5.5V for ATmega328P
- Temperature range:
  - Automotive temperature range: –40°C to +125°C
- Speed grade:
  - 0 to 8MHz at 2.7 to 5.5V (automotive temperature range: –40°C to +125°C)
  - 0 to 16MHz at 4.5 to 5.5V (automotive temperature range: –40°C to +125°C)
- Low power consumption
  - Active mode: 1.5mA at 3V - 4MHz
  - Power-down mode: 1µA at 3V

1. Pin Configurations

Figure 1-1. Pinout



1.1 Pin Descriptions

1.1.1 VCC

Digital supply voltage.

1.1.2 GND

Ground.

1.1.3 Port B (PB7:0) XTAL1/XTAL2/TOSC1/TOSC2

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Depending on the clock selection fuse settings, PB6 can be used as input to the inverting oscillator amplifier and input to the internal clock operating circuit.

Depending on the clock selection fuse settings, PB7 can be used as output from the inverting oscillator amplifier.

If the internal calibrated RC oscillator is used as chip clock source, PB7..6 is used as TOSC2..1 input for the asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

The various special features of port B are elaborated in Section 13.3.1 “Alternate Functions of Port B” on page 65 and Section 8. “System Clock and Clock Options” on page 24.

1.1.4 Port C (PC5:0)

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC5..0 output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

1.1.5 PC6/$\overline{\text{RESET}}$

If the RSTDISBL fuse is programmed, PC6 is used as an input pin. If the RSTDISBL fuse is unprogrammed, PC6 is used as a reset input. A low level on this pin for longer than the minimum pulse length will generate a reset, even if the clock is not running. The minimum pulse length is given in Table 28-4 on page 261. Shorter pulses are not guaranteed to generate a reset.

The various special features of port C are elaborated in Section 13.3.2 “Alternate Functions of Port C” on page 68.

1.1.6 Port D (PD7:0)

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, port D pins that are externally pulled low will source current if the pull-up resistors are activated. The port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

The various special features of port D are elaborated in Section 13.3.3 “Alternate Functions of Port D” on page 70.

1.1.7 AV_{CC}

AV_{CC} is the supply voltage pin for the A/D converter, PC3:0, and ADC7:6. It should be externally connected to V_{CC}, even if the ADC is not used. If the ADC is used, it should be connected to V_{CC} through a low-pass filter. Note that PC6..4 use digital supply voltage, V_{CC}.

1.1.8 AREF

AREF is the analog reference pin for the A/D converter.

1.1.9 ADC7:6 (TQFP and QFN/MLF Package Only)

In the TQFP and QFN/MLF package, ADC7:6 serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

1.2 Disclaimer

Typical values contained in this datasheet are based on simulations and characterization of actual ATmega328P AVR® microcontrollers manufactured on the typical process technology. automotive min and max values are based on characterization of actual ATmega328P AVR microcontrollers manufactured on the whole process excursion (corner run).

1.3 Automotive Quality Grade

The ATmega328P have been developed and manufactured according to the most stringent requirements of the international standard ISO-TS-16949. This data sheet contains limit values extracted from the results of extensive characterization (temperature and voltage). The quality and reliability of the ATmega328P have been verified during regular product qualification as per AEC-Q100 grade 1. As indicated in the ordering information paragraph, the products are available in only one temperature.

Table 1-1. Temperature Grade Identification for Automotive Products

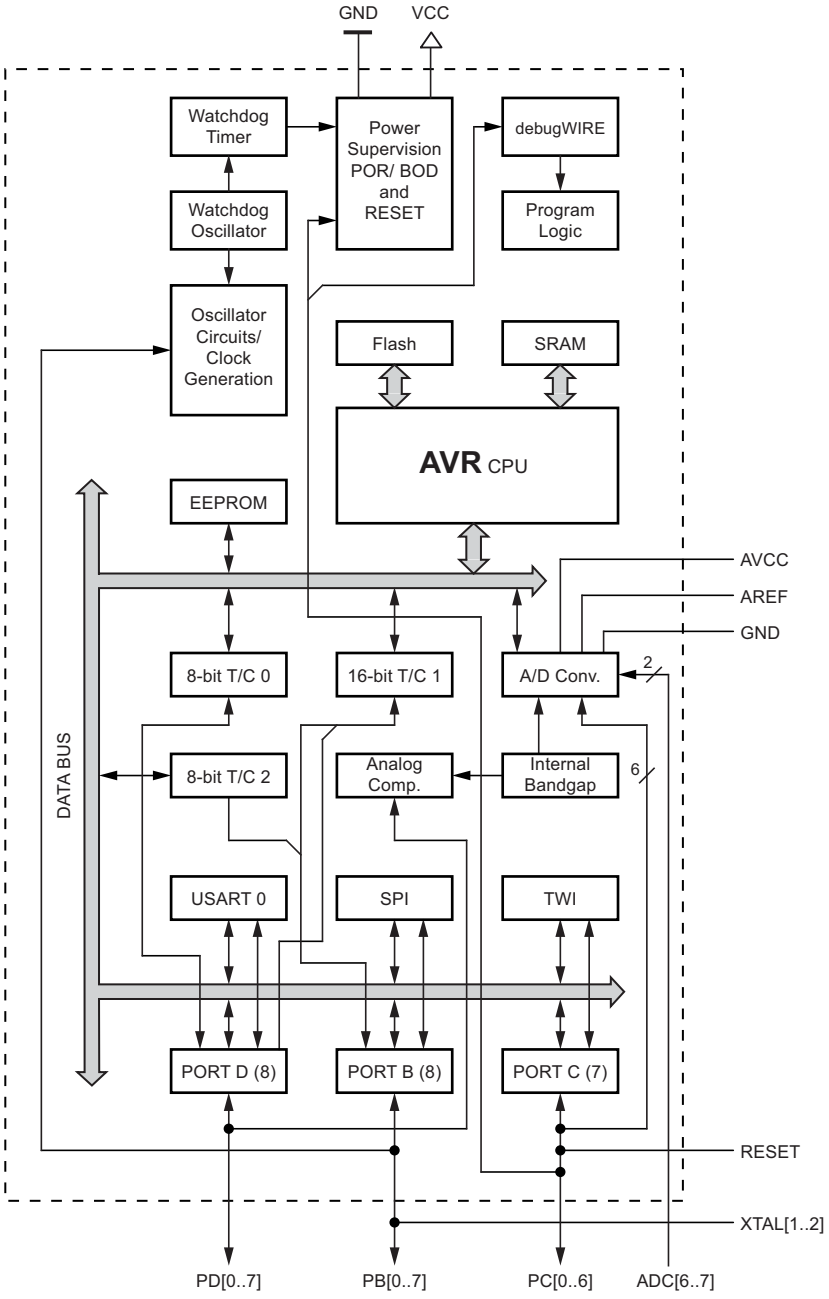
Temperature	Temperature Identifier	Comments
–40°C; +125°C	Z	Full automotive temperature range

2. Overview

The Atmel® ATmega328P is a low-power CMOS 8-bit microcontroller based on the AVR® enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega328P achieves throughputs approaching 1MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

2.1 Block Diagram

Figure 2-1. Block Diagram



The AVR® core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the arithmetic logic unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The Atmel® ATmega328P provides the following features: 32K bytes of in-system programmable flash with read-while-write capabilities, 1K bytes EEPROM, 2K bytes SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible Timer/Counters with compare modes, internal and external interrupts, a serial programmable USART, a byte-oriented 2-wire serial interface, an SPI serial port, a 6-channel 10-bit ADC (8 channels in TQFP and QFN/MLF packages), a programmable watchdog timer with internal oscillator, and five software selectable power saving modes. The idle mode stops the CPU while allowing the SRAM, Timer/Counters, USART, 2-wire serial interface, SPI port, and interrupt system to continue functioning. The power-down mode saves the register contents but freezes the oscillator, disabling all other chip functions until the next interrupt or hardware reset. In power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC noise reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC, to minimize switching noise during ADC conversions. In standby mode, the crystal/resonator oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption.

The device is manufactured using Atmel high density non-volatile memory technology. The on-chip ISP flash allows the program memory to be reprogrammed in-system through an SPI serial interface, by a conventional non-volatile memory programmer, or by an on-chip boot program running on the AVR core. The boot program can use any interface to download the application program in the application flash memory. Software in the boot flash section will continue to run while the application flash section is updated, providing true read-while-write operation. By combining an 8-bit RISC CPU with in-system self-programmable flash on a monolithic chip, the Atmel ATmega328P is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

The ATmega328P AVR is supported with a full suite of program and system development tools including: C compilers, macro assemblers, program debugger/simulators, in-circuit emulators, and evaluation kits.