New York Smart Greenhouse
NYSG

SMART GREENHOUSE

PROJECT MANUAL

Andrew Balogh
Jonathan Tran
Coleman Strine

August 2020

Contents

1. Introduction

1.1 Overview

The New York Smart Greenhouse (NYSG) is a miniature greenhouse with environmental monitoring and moderating functionality. The greenhouse is the perfect size to start seedlings or to grow a year-round herb garden. Keeping smaller succulents or cacti is also feasible.

The greenhouse is equipped with sensors to measure the internal temperature, humidity, soil moisture, and light levels. Depending on user-specified target levels, the greenhouse can adjust the environment to create optimal conditions for plant growth.

This manual will outline the procedure for recreating the greenhouse. It also includes documentation for the software and hardware. It is intended for high school students with limited to no experience with such a project.

Teaching box

Throughout the manual, boxes like this one will appear. While the main purpose of this manual is to detail the procedure to replicate the greenhouse, some important concepts will be expanded upon in greater detail to shed light on the design. The information in these boxes is at a very high level, intended to be a jumping off point for those curious to dive deeper into the engineering concepts behind the greenhouse.

1.2 Technical Features

The miniature greenhouse is constructed from a clear plastic storage bin. Depending on where the user will place the greenhouse, a clear lid may be desired as well. For this design, a bin approximately 24" long by 18" wide by 20" tall was used. A smaller container can be substituted, but bear in mind that a smaller enclosure will severely limit the plants that can be grown.

To control the greenhouse, a Raspberry Pi 4.0 B was used. The software was developed to run on a similar computer with GPIO pin functionality.

All software is written in Python 3.8. This manual is accompanied by a Jupyter Notebook, which is the main teaching tool for the programming aspect of this project. The software description in this manual will be limited to setting up the controller and running the system, not working with the software and developing code. For that, see the Jupyter Notebook.

For a complete Bill of Materials and Tools list, see Appendix A. For a compilation of reference images of the greenhouse in various stages of completion, see Gallery H.

1.3 Acknowledgements

A sincere thank you to Alfredo Iturralde, Meghan Daugherty, Brian Senchuk, Karissa Diggs, and Nick Solecky.

2. Greenhouse Structure

2.1 Greenhouse Container

Materials

- Storage bin
- Sharpie

Instructions

The greenhouse container is a modified storage bin. The sides, and preferably top, should be clear. This will allow light to enter for the plants. Keeping the plants inside this bin will allow us to control the growing environment.

1. *Determine greenhouse dimensions.* Depending on the size of the container used for the greenhouse, the exact dimensions and locations of elements may change. Most importantly, the soil depth and growing space must be determined. Other measurements will be given relative to this. Table 2.1 shows the recommended soil depth and growing space for a given container height, but these can be tailored to the user's specific uses. A container less than 12" is not recommended.

   Table 2.1

Container Height	Soil Depth	Growing Space
12"	5"	7"
13"	5"	8"
14"	5"	9"
15"	5"	10"
16"	5"	11"
17"	5"	12"
18"	6"	12"
19"	6"	13"
20"	6"	14"
21"	6"	15"
22"	6"	16"
23"	6"	17"
24"	6"	18"

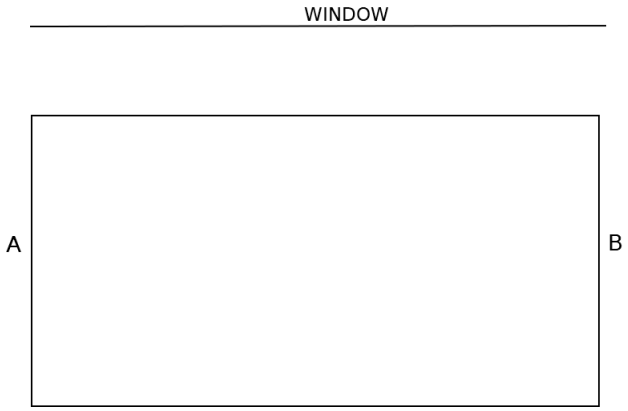
   Images throughout the rest of this manual show a container with 6" of soil and 14" of growing space.

2. *Mark soil depth.* Around the container, mark the determined soil depth with a marker.

3. *Determine orientation.* Choose which side to face towards the window. If your container has 4 relatively flat sides, choose a longer side to go towards the window. If your container only has 2 relatively flat sides, choose a curved side to go towards the window. On the two faces

adjacent to the one facing the window, label one “A” and the other “B” as in Figure 2.1. Power will be needed on side A, so if there is a preference as to which side of the window has power, mark that side A and the other B.

Figure 2.1: Greenhouse orientation



2.2 Ventilation

Materials

- Storage bin
- Sharpie
- Drill, 1/2” bit, 1-1/2” bit
- Utility knife

Instructions

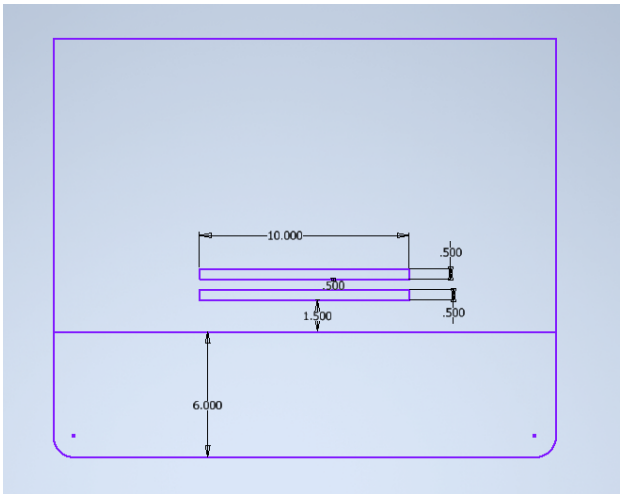
Ventilation in a greenhouse is useful to prevent overheating, create airflow, and replace CO2. This greenhouse will use a small fan and vents to create the ventilation. When the greenhouse is too warm, turning on the fan will suck the warm air at the top of the container out, and cooler air will be pulled in through the vents.

4. *Create vents.* On side B, measure up 1.5 inches from the soil line. Trace a 10 inch by 1/2 inch rectangle at the height measured, centered. From the top of that rectangle, measure up 1.2 inch and draw another rectangle of the same size.

   Use a 1/2” drill bit to create overlapping holes to remove the plastic from inside the rectangles. With a utility knife, clean up the edges.

   These are going to be the vents that serve as an air intake when the fan is spinning.

Figure 2.2: Vents



5. *Create fan hole.* On side A, measure down from the top 2 inches and make a small mark centered on the side. With the 1-1/2" bit, create a single hole. The fan will be mounted over this hole.

Air flow and static pressure

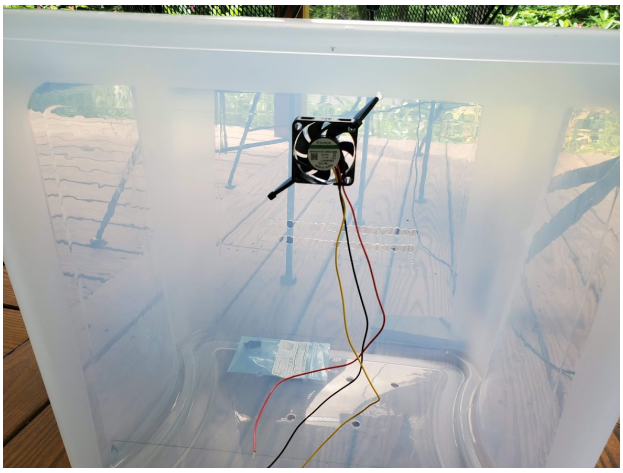
In a ventilation system like the one in this greenhouse, two measurements matter. Those are the air flow and the static pressure.

Air flow refers to the volume of air that a fan moves per time, usually measured in cubic feet per minute (CFM). The fan specifications will give a maximum CFM. If our greenhouse is 2ft×1.5ft×1ft (volume of $3ft^3$) and we want to replace the air in the greenhouse every 1.5 minutes, the fan needs a CFM of $3ft^3/1.5min$=2CFM.

Static pressure is the pressure that the fan is blowing against. If the fan is trying to blow air into a box with no vents, the air pressure will build up inside the box. At some point, the static pressure will be too high for the fan to blow air into the box. This is why we added large vents to the back of the greenhouse. The pressure in a system will be determined by the number and size of vents and other objects between the fan and vents that resist the flow of air.

6. *Mount fan.* Hold the fan centered on the hole and mark where the top right and bottom left mounting holes are. Drill a 1/8" hole at those two marks. It will be close to the larger hole, but it is okay if it connects as long as there is a clear notch where the 1/8" hole is. About an inch away from each of those holes diagonally, drill another hole. Take two small zip ties and mount the fan on the outside of the greenhouse using the holes created. Make sure the wires are on the bottom left of the fan so the sticker is upside down, and the sticker is on the outside so that the fan sucks air out of the greenhouse.

Figure 2.3: The mounted fan.



2.3 Water System

Materials

- Storage bin
- Solenoid valve
- 2L bottle
- Adapters
- Zip ties
- Tubing
- Drill, 5/8” drill bit, 1/8” drill bit
- Utility knife

Instructions

Any indoor plant needs to be watered manually. This greenhouse will include a 2L water reservoir that will be used to water the plants. A solenoid valve will control the flow of water and a soaker hose will efficiently deliver water to the roots of the plants. A soaker hose is porous and allows water to leak out along its length. Once watered, the reservoir will need to be refilled.

7. *Create drainage holes.* On the bottom of the greenhouse, make a number of holes with the 5/8” bit. There should be about 5 holes per square foot. For a bin of 24” by 18”, 5 holes/ft^2 $\times$ 2ft $\times$ 1.5 ft = 15 holes.

   The water applied to the plants will drain out of these holes. Not providing a way for water to escape will cause the roots of the plants to sit in water and drown or get root rot. Providing good drainage will keep the plants’ roots healthy.

Drainage

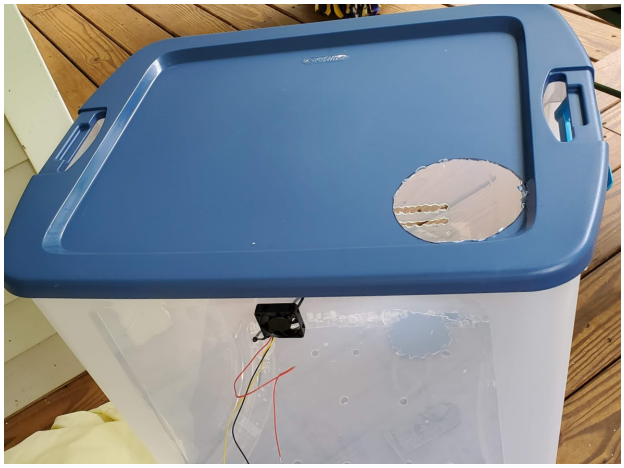
When keeping plants in containers, one of the most important considerations is drainage. Most plants do not like to be kept in constantly wet soil. Doing so can cause the plant to drown or contract root rot. Plants should be watered only once the soil dries out, and kept in containers with adequate drainage.

The best way to provide drainage is to put holes in the bottom of the container. Do not use gravel or similar materials in the bottom of your container, this actually is worse. Water moves poorly between different materials, so adding stones to the bottom creates what is called a perched water table. The water level in the soil will be higher with stones at the bottom, letting the plants sit in soggy soil.

To prevent soil from exiting the holes or the holes getting clogged, we will line the bottom of the bin with old fabric from a pillowcase. Other fabrics, or materials like coffee filters, peat moss, or coco coir, can also be used.

8. *Create reservoir opening.* Take the top of the bin. Place the lid so that you are looking at it in landscape orientation. On the right hand side of the bin, in the corner closest to you, trace out a 4 inch circle. If the lid is lipped, the circle can be in the thin part, as long as it is no more than an inch and a half from the edge. Then, use a utility knife to cut out the hole.

Figure 2.4: The reservoir opening.



9. *Drill reservoir container.* Take the soda bottle. With a 5/8" drill bit, drill a hole in the bottom of the bottle.

10. *Mount solenoid valve.* Centered on side A, mark two points an inch and a half up from the soil level and one inch apart. Use the 1/8" drill bit to make two small holes. On the right hand side (while facing side A) make another small hole up and to the right of the rightmost hole, about a half inch in each direction. Use the first two first holes to zip tie the valve to the wall (black coil up, arrow facing away from the reservoir). The wires should then be on the right and can be pulled through the third hole.

Water pressure

Water pressure refers to how forceful the flow of water in a system is. Most residential or commercial water system, like those in your house, have a water tank that maintains a constant water pressure. When you turn on your faucet, the water pressure is what causes water to come out at a constant speed. Water pressure is usually measured in psi, pounds per square inch.

Our water system in this greenhouse is gravity-fed, so it makes sense to talk about the pressure in terms of pressure head. Head refers to the height of the water that would create the pressure felt, expressed in feet or meters. 1 foot of head = .43psi. Since the soda bottle is about 1ft tall, the head at the point of the hose is 1ft, or .43psi. As water leaves the system when the valve is open, the water pressure will lower.

11. *Connect with tubing.* Attach the connectors to the bottle and the intake side of the valve. Use Teflon tape on the threaded connections to prevent water from leaking. Cut vinyl tubing long enough that the reservoir is smoothly connected to the valve with no kinks and short enough that the tubing does not dip below the level of the valve, when the reservoir is in place through the hole. Use the barbs to connect the hose. Then, use a large zip tie to mount the reservoir. If necessary, prop out the reservoir.

Figure 2.5: Water reservoir

12. *Connect soaker hose.* Thread the soaker hose adapter with the NPT die. This can be hand threaded. Then screw it into the valve, again using Teflon tape. Cut soaker hose to desired length to reach all plants and cut it. Attach it to the barbed end of the adapter and plug the other end. Place as desired and use the staples to hold it in shape. If you find that the soaker hose is not emitting water where you want, or is emitting too slowly, you can manually poke holes in the hose as desired.

2.4 Lights

Materials

- LED strip
- 2-conductor wires
- 10mm LED connector
- Zip ties
- Drill, 1/8" bit
- Wire cutters
- Optional - skylight
- Packing tape
- Utility knife

Instructions

Plants use photosynthesis to turn sunlight into food, so the greenhouse needs to ensure the plants have proper light levels. The lights used here are optimized to promote plant growth and flowering. They are called plant grow lights.

13. *Cut strips of lights.* Take the reel of lights and observe that there are repeated pattern. Cut a strip of lights long enough to wrap around side A through the side away from the window and across side B. Make sure your cut on the markings.

14. *Attach leads.* Use the 10mm LED connectors to attach two-conductor wire to one end of the lights. If you plan to use a breadboard, make the leads 8 inches. If you are using a PCB, make the leads 18 inches. Make sure the red wire is connected to the positive terminal of the lights and the white to the negative. The wires do not need to be stripped before being put in the LED connector.

Wire Color Conventions

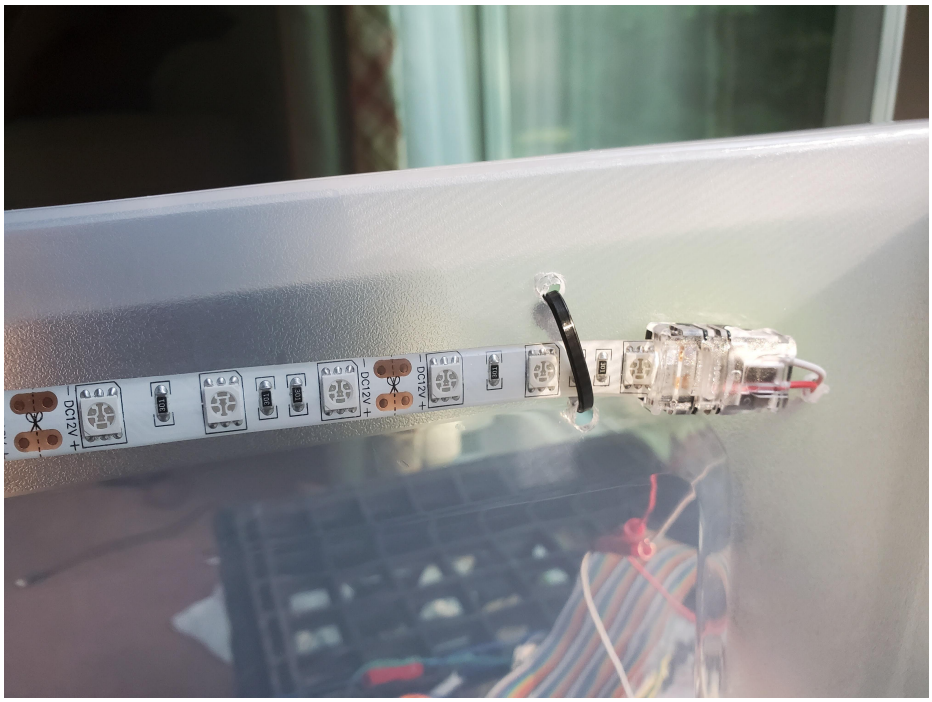
These instructions specify which wire needs to be connected to which pad on the lights. There is no physical difference between the 2 wires, so why is it important? In all professions, there are conventions that are followed to make sure that everyone is on the same page.

In electronics, one of the conventions is wire color. Different colored wires are meant for different things, even though there is no physical reason for that to be the case. Here, our lights are powered by a 12V supply, meaning they have 12V across them to work properly. For LEDs, it is very important that the positive side of the voltage is applied on one terminal and the negative side on the other terminal. Many other circuits also need to specify which terminal is positive of negative. Later, we see that the lights and the fan both have specific wires for positive and negative voltages, while the valve and the heater can be wired either way.

In a DC circuit with 2 terminals, like the lights and the fan, people who work with electronics designate a positive supply wire with red and a negative supply wire with black. If someone picks up the fan and doesn't have the datasheet with it, they can be confident that if they apply positive voltage on the red wire and negative on the black, the fan will work!

15. *Mount the lights.* Remove the adhesive backing from the lights and, an inch from the top of the bin, stick them on the three sides. Make sure the wire leads are in the corner on side A by the window. Drill small 1/8" holes periodically along the lights and zip tie them for extra support. Drill an additional hole for the leads to stick out to connect to the controller.

Figure 2.6: The mounted lights on side A



Plants and light

Plants evolved to grow under sunlight, where they take sunlight and use it as energy to create the food it needs to grow. Sunlight contains all wavelengths of visible light, but plants only use certain wavelengths. They have photoreceptors (proteins that react to light). Chlorophyll is the most commonly referenced photoreceptor important to photosynthesis. Chlorophyll and other similar proteins best absorb blue and red light. So, we can use red and blue LEDs to provide light to the plants and let them grow.

How much light do plants need, though? In agriculture and gardening, plants are put into categories based on how long they need direct sunlight. Direct sunlight is when the plant is receiving sunlight directly, and is not in shade. Usually, the plants will be placed in the following categories:

- Full sun: 6+ hours of direct sunlight
- Part sun: 3-6 hours of direct sunlight
- Part shade: 3-6 hours of direct morning sunlight
- Full shade: Less than 3 hours of direct sunlight

16. *Optional - Create skylight.* I used a storage bin with a blue top, which blocked a lot of the light coming from the window. To remedy this, I created a skylight. Take something clear, like the top to a takeout tray or even just plastic wrap, that is large enough to let a lot of light in. Cut out the shape in the lid of the greenhouse and use clear packing tape to tape down the sides.

Figure 2.7: The skylight

2.5 Heating Element

Materials

- PTC element
- 2-conductor wire
- Stone
- Wire cutters
- Butt splices

Instructions

The greenhouse includes a small heating element for cases where the greenhouse is placed outside, or in a place that is not normally heated. It will keep plants healthy through cold temperatures or prolong the growing season, perhaps indefinitely.

17. *Attach wires to heater.* Just as with the lights, slightly separate and strip a the end of the two-conductor wire. Use a butt splice, dolphin, or solder to connect the leads of the heater to the wires.
18. *Make connection.* The wires need to be long enough two connect the heater, placed in the greenhouse by the vents, to the controller on side A by the window. If you plan to use the provided PCB for the circuitry, cut the wire long enough to reach the board. If you plan to use a breadboard, cut the wire a few inches short and follow the step above to attach 2 breadboard jumper cables to the wires. For added security in the connection, wrap in electrical tape.
19. *Place heater.* Place the heater on a medium sized rock to act as a heat sink.

2.6 Sensors

Materials

- Sensors
- 3- and 4-conductor wire
- Drill, 1/8” bit

Instructions

The sensors are what collect data on the greenhouse environment. They are connected electrically to the controller, so the controller can read the data on them and report it to you.

20. *Connect temperature and humidity sensor, light sensor.* These sensors have 4 holes on them. If you are using a breadboard, use a 4-pin JST PH connector and solder them to the holes. Use red for V+, black for GND, yellow for SDA, and white for SCL. Then connect gently insert the jumper cables into the female pins. If using a PCB, simply solder 4-conductor wire about 10” long.
21. *Connect soil moisture sensor.* The soil sensor comes with connectors. Insert the connector.

22. *Place sensors.* Drill a 3/8" hole on side A to insert the wire leads through. Place the sensors as desired. Recommended: place the soil moisture sensor close the soaker hose, the temperature sensor about halfway up the container when full of soil, and the light sensor along a wall representative of the light plants get. The leads can be as long as needed for the sensors to connect to the board.

2.7 Soil

Materials

- Soil
- Fabric
- Tray
- Scissors

Instructions

The greenhouse needs soil in which to grow plants. These steps can be done as soon as you feel comfortable with where the greenhouse will be placed permanently and want to place it there, continuing all work on it in its new home.

23. *Cut pillowcase.* Cut a pillowcase or other fabric to the size of the bottom of the bin. Lay it down in the container.

24. *Drainage catcher.* Place the greenhouse on a large drip tray, or cut a garbage bag to lay down under it. The water put into the soil will drain out of the holes, so you want some way to catch that water or it will spill all over the floor. Prop the greenhouse up about an inch over whatever you use to catch water.

25. *Fill with soil.* Add soil to the greenhouse up to the line you drew when labeling the sides. Be careful not to get soil in other elements of the greenhouse that are already set up. Now, the soaker hose and soil moisture sensor can be placed in the soil.

3. Electronics

3.1 Circuit Introduction

Simply, a circuit is a path for electricity to flow. We can design circuits that manipulate or respond to certain important electrical quantities to achieve a certain electrical goal.

Voltage, Current, and Power

Current is an electrical quantity refers to the rate of the flow of positive charge. In a circuit, electrons are what move through the wire. However, since electrons are negatively charge, current is defined to be opposite to the flow of electrons. Current is measured in amps (A).

Voltage is the force that causes electrons to flow. It is the difference in energy for a charged particle between two points. Voltage is measured in volts (V).

Voltage and current can be conceptualized by thinking of a stream. The force of gravity causes water to flow downhill, to where there is lower potential energy. The current is the water flowing through the stream.

The other important quantity is power. Power is the energy per unit time, usually measured in Watts (W). Power in a circuit is equal to the voltage times the current ($P = VI$).

Voltage, current, power can either be AC (alternating current) or DC (direct current). In a DC circuit, the quantities are all constant. In an AC circuit, the quantities oscillate as a sine wave. While the power coming from a wall outlet is AC, the circuits in this greenhouse are all DC.

The circuits are comprised of components each of which behave differently and have characteristic responses or alterations of voltage, current and power. In designing a circuit, the designer carefully chooses the components that they want and connects them with wires. The wires are simply the conduit for electricity and ideally have no impact on voltage, current, or power.

Electronic Components

There are 5 main electronic components you will commonly find in circuits, and each has different characteristics and uses.

The simplest electronic component is the resistor. The resistor is governed by Ohm's Law, which states that the voltage across the resistor is proportional to the current through it. Simply, $V = IR$ where R is the resistance of the resistor. Resistance is measured in Ohms, although practically most resistors used are in the kiloohm (1,000 Ohms) or greater range.

The next component is the capacitor. The capacitor is a component that resists a change in voltage across it. Capacitance is measured in Farads (F), and is usually very small (microFarads or smaller). Analysis of capacitance in a circuit requires the use of calculus.

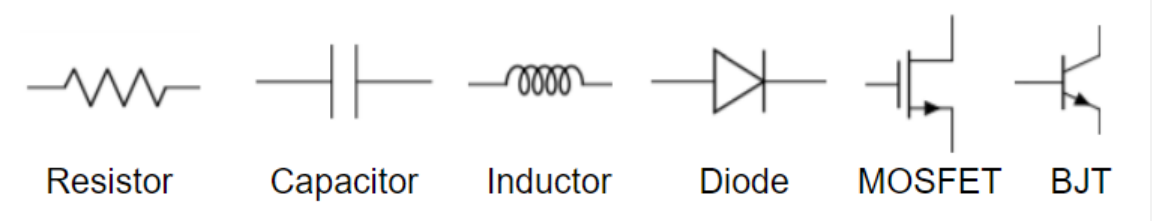
An inductor is very similar to a capacitor, but instead of resisting a voltage change it resists a current change. Inductors are measured in Henrys (H). Again, inductors are harder to analyze than a resistor.

A diode is a component that, very simply, allows current to flow in only one direction. The diode has two terminals, the anode and the cathode. If the voltage on the anode is higher than the voltage on the cathode, then current will flow. Current will never flow from the cathode to the anode.

The final component is the transistor. Transistors are the main component in modern electronics. Computers, your phone, etc. are all made up of millions of transistors. Transistors act as an electrical switch. Unlike the other components, transistors have three terminals. There are two types of transistors.

The first type of transistor is the MOSFET transistor. The MOSFET has 3 terminals, the gate, the source, and the drain. If the voltage on the gate is high enough, then current will flow from the source to the drain. If the voltage is too low, no current will flow.

The second type of transistor is the BJT transistor. The BJT also has 3 terminals, the base, collector, and emitter. It functions very similar to the MOSFET, but is controlled by current instead of voltage. If enough current is flowing into the base, then current will flow from the collector to the emitter.



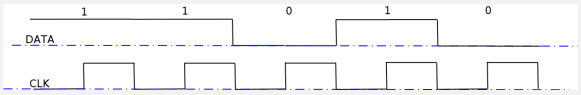
One of the most important uses of a circuit is to send electrical signals. The signals carry information, and are how electronics can communicate. In the greenhouse, we want the computer to communicate with the sensors and the peripherals. We place circuits between the electronics we want to communicate. Such a circuit can be as simple as a few wires and no components, whereas some need loads of components.

Electrical Signals

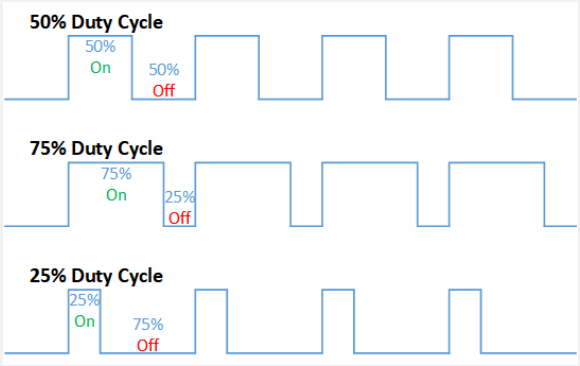
The Raspberry Pi controller has a 40 pin GPIO header. GPIO stands for General Purpose Input/Output. On most of the pins, the voltage can be changed or measured. These GPIO pins are how the controller communicates with the sensors and the peripherals.

Electrical signals are sent in binary, or base-2. Our normal number system is base-10. Instead of having 10 different symbols, 0-9, to represent our numbers, in binary everything is represented by either a 0 or a 1 (each 0 or 1 is called a bit). On the GPIO pins, the voltage is either a 1 (3.3V) or a 0 (0V). Information can be transmitted using this binary protocol using only 2 connections, a data line and a clock line. The data line has the information as a series of 0s and 1s, and the clock line tells when to take the measurement for each bit.

For example, if a sensor wanted to send the number 26 (0b11010) to the controller, it would send the info on the lines like in the following diagram. Every time the clock line rises (goes from low to high) the bit will be read. On the first edge, the data will have a 1. It will stay 1 for the second rising edge. Then, it will change to 0 before the third rising edge then 1 again, then 0. In this way, data can be sent on two lines.



The control of the peripherals is a little bit simpler. If the GPIO pin is set to 3.3V, it will turn on the peripheral. If the pin is set to 0, the peripheral will turn off. The power delivered to each peripheral can be changed by sending the signal with a varying duty cycle. A duty cycle of 100 means the signal is always high and delivers the most power. A duty cycle of 0 means the signal is always low and delivers no power. With a duty cycle of 25%, the fan will spin slower than for a duty cycle of 75%.



The signals described above are all *digital* signals, meaning they take prescribed, binary values (0 or 1) in transmitting data. There are also *analog* signals, in which the voltage scales continuously. The soil moisture sensor sends an analog signal, where a higher voltage output means the soil has more moisture. You can convert between analog and digital signals using an *analog to digital converter (ADC)* or *digital to analog converter (DAC)*.

The most common methods to create circuits in general are using a breadboard or using a printed circuit board (PCB).This manual details how to make the necessary circuitry on both (the Gerber file to create the PCB is on the GitHub).

3.2 Breadboard

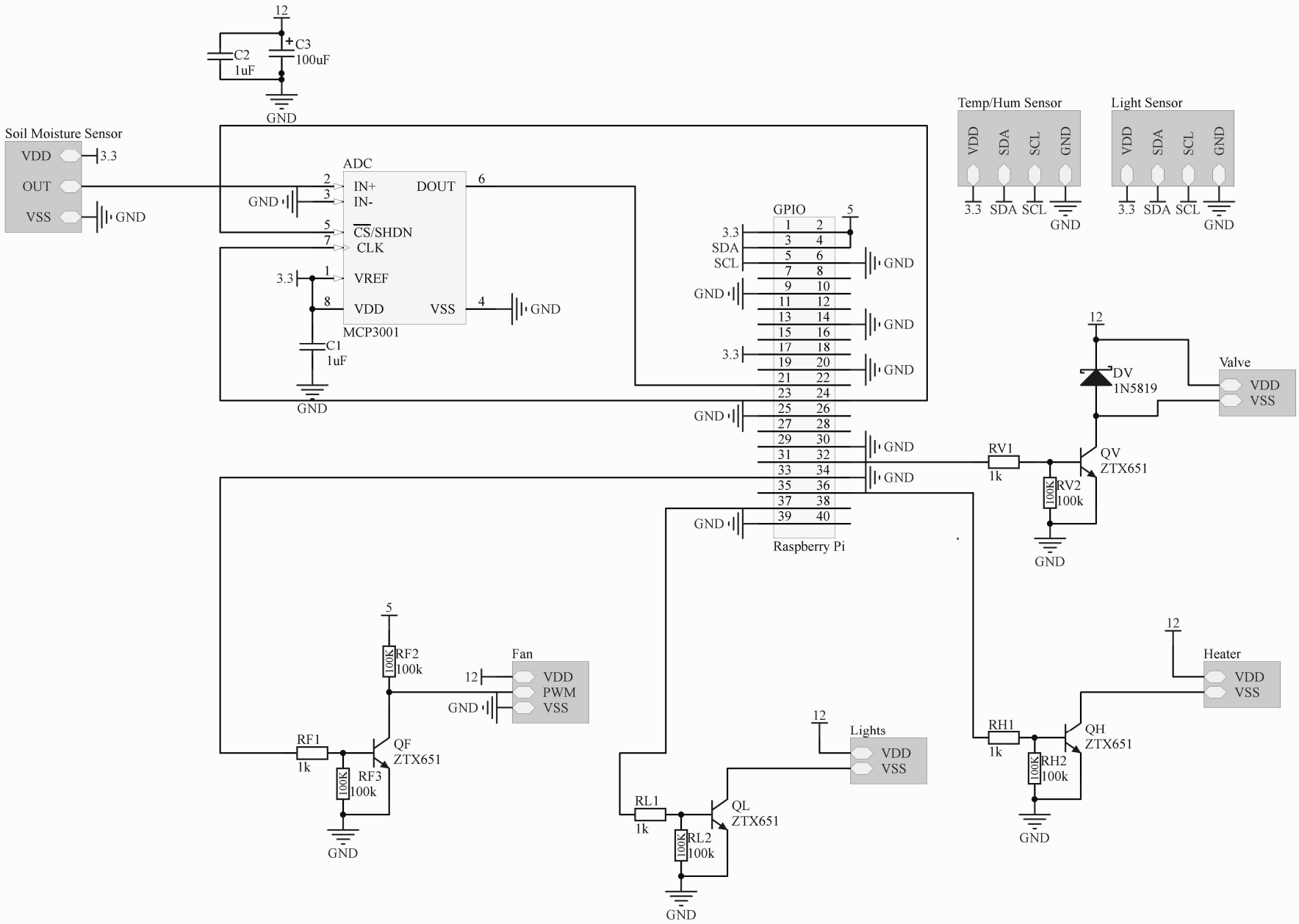
The breadboard is a plastic board with holes that allow us to place components in the holes and use jumper wires to connect them. The advantages of a breadboard is that they are easy to change and do not need to be custom built. That is, any circuit with the right components can be built on a breadboard. However, they can be more complicated to look at build due the number of jumper wires, and certain components can not be easily used with a breadboard.

Breadboard basics

1. The outside power rails are connected vertically.
2. The numbered rows are connected horizontally.
3. The deep trench in the middle separates connections, allowing us to use certain components and access each pin individually.

To create the breadboard circuit needed to run the greenhouse, use the circuit diagram on the next page. Breadboards are very flexible, so I won't tell you exactly how to set it up. As long as the circuit is the same it will work no matter where you put the individual components. However, I will offer a few guidelines.

- Use the red expansion board to connect to the GPIO pins. The pins sticking out of it can be placed on the large breadboard. Make sure the pins are on separate sides of the deep trough.
- The ADC can also be placed on either side of the deep trough.
- Use the jumper cables to connect nets that need to be connected.
- When connecting the ribbon cable, double check that it is on the right way. Use a multimeter to test for a short circuit between the pins you think are connected on the Raspberry Pi and breadboard. If they are shorted, then the orientation is correct.
- The peripheral elements may need to be connected to leads that can plug into the breadboard. Use extra jumper cables and connect them to the leads with dolphins, butt splices, or solder and electrical tape/heat shrink. Then plug into the breadboard.
- Use one breadboard for the peripherals and one for the sensors. That way you can have one at 12V and the other at 3.3V.
- Connect the ground of the Raspberry Pi to the ground of the peripherals and sensors.
- The solenoid valve may come with a power jack to terminal block connector. If so, use this to power the breadboard by putting jumper cables into the terminal block and tightening the screws. Otherwise, solder connectors to the other power jack.
- Check the datasheet for the transistor and for the ADC, and potentially some other elements, to determine which pins are which. Datasheets are the documentation provided by the manufacturer of electronic components outlining their specifications for engineers to use.



Title	Smart Greenhouse Interface	
Size: A	Number:	Revision:
Date: 8/17/2020		Sheet of
File: breadboard_schem.SchDoc		Drawn By: NYSG

3.3 PCB

A PCB is a manufactured board that is a layer of a plastic-like material sandwiched between two layers of thin copper. The copper layers act as the wire connecting components. The advantage of a PCB is that it is custom built for a particular circuit, and any component can be used with one. They can be easier to use. However, the disadvantages are that they are not flexible once they are printed (it is hard to change things) and they require the components to be soldered on for the proper connections.

Through-hole vs Surface-mount

Components come in two styles: through-hole or surface-mount. Through-hole components have long terminals that are placed through a hole in a PCB then soldered. These components can be used in a breadboard as well. Surface mount components are placed on a pad on one side of a PCB and soldered. These cannot be used in a breadboard. In this circuit, the transistors, resistors, diodes are all through-hole while the capacitors are surface-mount.

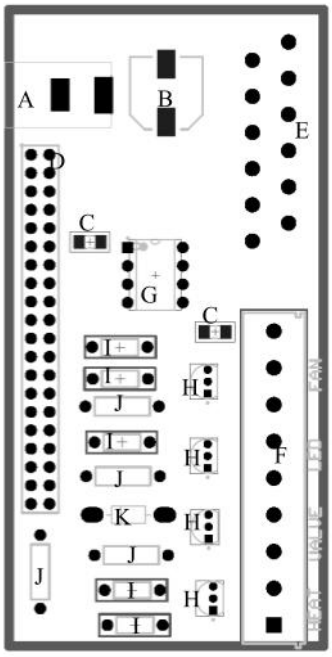
Using a PCB is easier than a breadboard since all the connections are fabricated as part of the board, but it requires soldering. To use the PCB, solder the components in based on the assembly diagram on the next page. While soldering, make sure to make good connections, not cold connections. If the solder is in a ball, instead of a smooth curve around the component connection, you probably have a cold solder and need to improve the connections.

Use a multimeter to test for shorts before applying power.

Once the peripherals and sensors are in their proper place, connect the peripheral and sensor leads into the appropriate terminal blocks and tighten the connection with a screwdriver if necessary. Any wire of the right size (18-22 AWG) can be connected securely). If any leads are too short, use wire and connectors (dolphins, solder and wrap, butt splices) to lengthen them.

Figure 3.1 shows the assembly drawing for the PCB, with the components labelled.

Figure 3.1: PCB assembly diagram

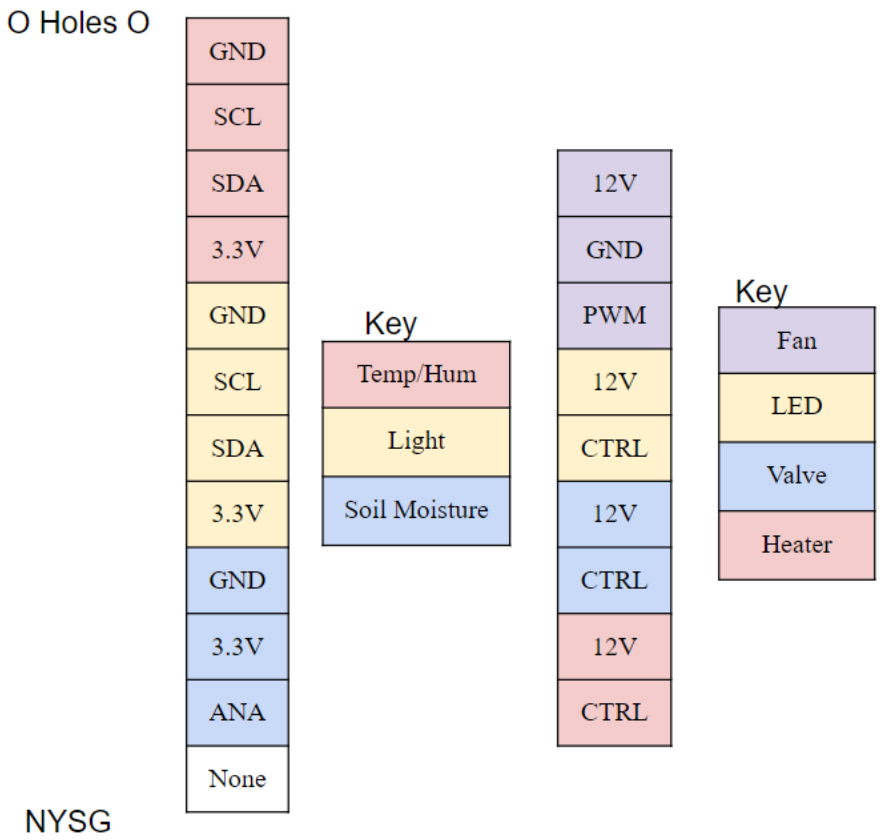


Designator	Quantity	Description	MPN
A	1	Power Jack	PJ-037AH
B	1	100uF capacitor	EEE-HBH101UAP
C	2	1uF capacitor	C1206C105Z4VACTU
D	1	40 position header	0010897402
E	1	12 pos terminal. block	1990106
F	1	9 position terminal block	282837-9
G	1	Analog to digital converter	MCP3001-I/P
H	4	NPN	ZTX651
I	5	100k resistor	CFR-25JB-52-100K
J	4	1k resistor	CFR-25JB-52-1K
K	1	Schottky diode	1N5819-T

Finally, Figure 3.2 shows the relative connections for sensors and peripherals for the 12 positions and 9 position terminal blocks. The holes denote the two large ground vias and the NYSG denotes

the text “NYSG” on the PCB itself, for orientation purposes.

Figure 3.2: Terminal block pinouts



4. Software

4.1 Raspberry Pi Setup

The controller for the greenhouse is the Raspberry Pi 4.0 B.

26. *Required Items.*
    (a) Raspberry Pi 4 Model B with 2 GB RAM
    (b) Transcend brand name UHS-I 8GB Micro SD and micro SD card holder
    (c) Power supply cable for Raspberry Pi
    (d) Laptop/desktop computer with SD port
    (e) Reliable network connection via WiFi
27. *Preliminaries.*
    (a) Check that you have a desktop computer with a full-sized SD card port
    (b) Check that the desktop has a WiFi connection
    (c) Verify you have an Raspberry Pi 4 B (2GB RAM memory)
    (d) Check you have a 8 GB micro SD card (small square thin card) and the associated larger full-sized SD Card adapter
    (e) Check you have a 3 Ampere power supply cable for the Raspberry Pi
    (f) Check you have the three heat sinks (these are small pieces)
28. *Flash micro SD card..* While the Raspberry Pi is a computer, it does not come with a pre-installed main memory. Instead, it needs its OS to be flashed to the micro SD card, which is then inserted into the micro SD card port on the bottom of the board. We have created a custom image specifically for this project, which you can flash directly onto your SD card. This image contains the Raspberry Pi OS, as well as all files and configurations necessary to get you started with this project. The .img file that you will need can be found at `https://github.com/colestrine/XXXXXXX.git`. Next, visit `https://www.raspberrypi.org/downloads/` and download the Raspberry Pi Imager. This will allow us to flash the .img file to the SD card.

Memory

On our Raspberry Pi (and in all general computers), there are two primary types of memory - Main Memory and RAM (Random-Access Memory). As far as our Raspberry Pi is concerned, main memory consists of our SD card that we plug into the bottom of the board, and RAM is built into chips that are pre-installed on the top of the board.

Main Memory Main memory is a fuzzy term that refers to long-term, non-volatile memory (holds its contents when the system is powered off). Most often, "main memory" refers to a Hard Disk Drive (HDD) - the main storage space in a PC. However, main memory can also consist of a USB drive, an SD card, or any other long-term storage medium. In the case of a Raspberry Pi, the main memory is the filesystem on the microSD card.

RAM Random-Access Memory (RAM) is the portion of memory on a computer that the CPU can directly interact with during processing. When an application loads, it pushes a certain amount of information to the RAM for consumption by the CPU. The CPU can interact with the application by pushing data back to RAM. This process exists because RAM is much faster than main memory (below), and so by forcing the CPU to interact mostly with RAM (as opposed to main memory), things can be sped up. RAM is intended for only short term storage while a process interacts with a CPU, as RAM is "volatile", or is cleared when the system is powered off. As the RAM will not hold its contents when power is lost, we do not store any files or long-term data here - these are stored in main memory.

29. *Run the Raspbery Pi Imager.* This program makes the flashing easy.
    (a) Load the micro SD into the larger full-sized SD card
    (b) Load the full sized SD card into the desktop's SD card port
    (c) Download Raspberry Pi Imager for your desktop computer. This should be for the specific OS that you have, for example, MAC OS or Windows.
    (d) *Choose the Image File.* Choose the .img file that you downloaded earlier
    (e) *Choose the SD card.* Choose the SD card that the Raspberry Pi will run from. Any data on the card will be deleted.
    (f) *Write.* Select Write and let the flasher run. This will take a few minutes.
    (g) *Finish.* When your system is done flashing, do not remove the SD card from your computer, you will need it for the next step.

30. *Configure Your OS.* This will connect your Raspberry Pi to your local network and enable SSH.
    (a) With your SD card still plugged into your computer, use your computer's file explorer to navigate to the SD card's boot directory
    (b) Create a new file in the boot directory called "wpa_supplicant.conf"
    (c) Use Notepad (Windows) or TextEdit (Mac) to edit the file
    (d) Copy and paste the following text into the file:

```
ctrl_interface=DIR=/var/run/wpa_supplicant
GROUP=netdev
update_config=1
country=US
```

```
network={
    ssid="Name of your wireless LAN"
    psk="Password for your wireless LAN"
}
```

Replace “Name of your wireless LAN” with the SSID (name) of your WiFi network (leave the quotes). Replace “Password for your wireless LAN” with the PSK (password) of your WiFi network (leave the quotes).

(e) Create a new file in the boot directory called “ssh” (no file extension). Do not enter any contents into this file. This will enable SSH on your Raspberry Pi.

(f) Now, you can unplug your SD card from your computer, and insert it into your Raspberry Pi.

Operating Systems

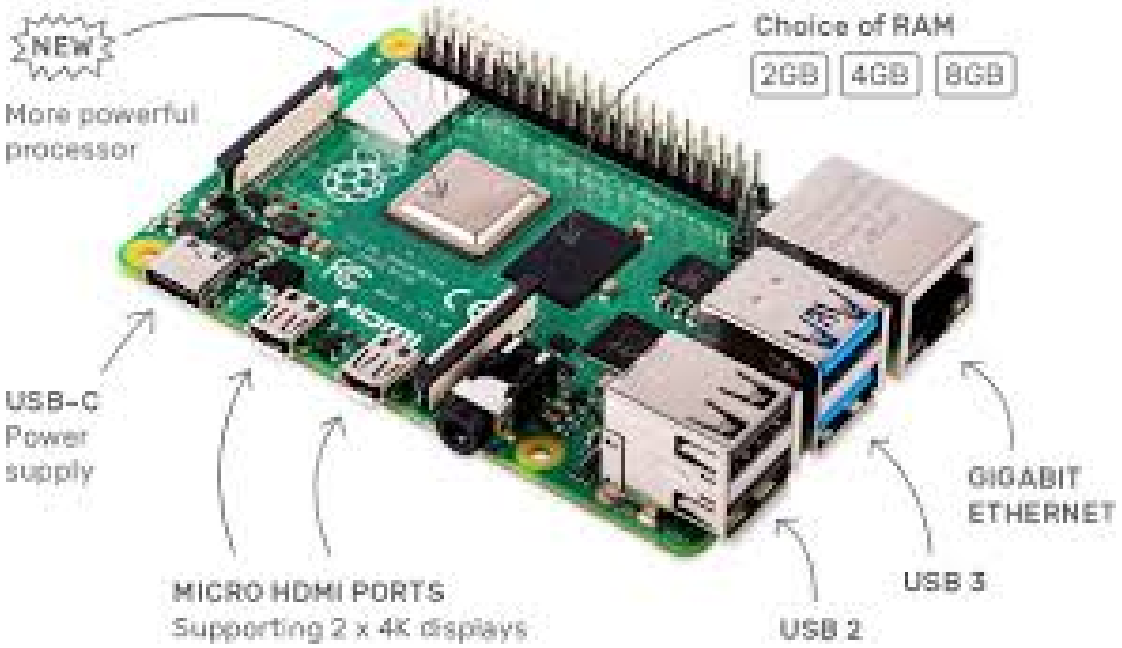
The operating system (OS) is the most fundamental piece of software on a computer. It controls and manages all other actions that the computer performs, such as input/output (I/O) operations, memory operations, and the coordination of application processes. I/O operations include things such as reading input from a keyboard as you type, or changing the color of the pixels on the screen while you watch Netflix. Memory operations consist mainly of reading and writing to memory, and happen when you save a Word document to your Documents folder (writing to memory), or open a PDF from your Desktop (reading from memory). The OS also coordinates or "schedules" all of the processes which are running at any given time. This process involves designating blocks of time during which each running process gets to use the CPU (the processor). If you are watching Netflix while you are writing a paper on Microsoft Word, it appears as if both applications (Netflix and Word) are running simultaneously. In a computer with a single-core CPU, however, this is actually not the case - in reality, the two programs are running "concurrently", which means that the CPU is only processing them one at a time, but switching back and forth so quickly that the human eye can not tell.

To make all of this possible from the programmer's perspective, the OS also provides what is called an Application Programming Interface, or "API". The API provides high-level abstractions that a programmer can use to leverage the abilities of the OS. For example, if a programmer is writing a program that needs to read a file from memory, they do not need to write the code that examines and interprets every single bit stored in memory - if they did, much of their time and energy would be spent on relatively un-interesting functionality. Instead, they can leverage the API provided by the OS to do this for them. In fact, neither the programmer nor any program that they write can ever interact directly with hardware or memory - all of these operations must be done through the API. Essentially, the programmer must use the API to "ask" the OS to get the contents of the file for them. In practice, the API is further abstracted through high-level programming languages, such as Python. For example, in Python, there exists a method called read(), which will read and return the contents of a file. Having the ability to call that single method is infinitely more productive than if the programmer had to write a program to manually search through main memory every time they wanted to retrieve information. If this concept is extended to all of the tasks that are handled by the OS, such as adjusting individual pixel colors and brightnesses, handling keyboard and mouse input, and managing processes, the increase in simplicity and productivity from the programmer's perspective can truly be appreciated. When the programmer is able to rely on the language and API to handle all of the low-level tasks, and they can simply call a high-level method such as read(), then they are then free to focus on all of the interesting work that makes our applications, such as Word or Netflix so useful (and distracting).

Also note that "operating system" is a very general term and refers to many different systems. The two most widely-known examples of operating systems are Windows and MacOS, which are both PC OS's - however, these are far from the only examples of operating systems in the real-world. There are, in fact, many different PC operating systems (including Linux, of which there are dozens of popular varieties), as well as operating systems which run on devices other than PC's, such as phones (iPhones run iOS and Android phones run AndroidOS), TV's (Roku TV's run RokuOS, Android TV's run a version of AndroidOS called Android TV OS, Amazon Fire TV's run FireOS), and other smart home devices (Amazon Echo's runs FireOS).

31. *Plugging Wires into Raspberry Pi.* The Raspberry Pi features a number of connections for input and output devices, including keyboard/mouse, display, and charger.

Figure 4.1: The Raspberry Pi 4 Model B.



(a) Look carefully at Figure 4.1, and notice where the different ports are located relative to each other. You will not need to worry about the Gigabit Ethernet port.

(b) Begin by plugging in the USB-C power supply into the USB-C power supply port on the Raspberry Pi and a wall outlet. A red light and green light should blink on the pi, next to the power supply on the Raspberry Pi.

32. *Enabling SSH on your local device to allow work on Raspberry Pi.*

(a) For Reference: Open `https://maker.pro/raspberry-pi/tutorial/raspberry-pi-4-ssh-wi-fi-`

(b) If you have a MAC, you are done. Macs have SSH enabled by default

(c) If you have a Linux, Windows or any other computer, follow the below instructions

(d) On your local device, go to `https://putty.org/`. Download PuTTY in the first "here" button.

(e) Once PuTTY is open, click on default settings.

(f) Click on PORT and enter 22

(g) For Connection type, click the "SSH" radio button

(h) In the box "Host Name (or IP Address)", type 'raspberrypi'.

(i) Click Save on PuTTY

(j) Click Open to open a PuTTY session

(k) Note: If PuTTY can not find the device, try using 'raspberrypi.local' as the host name. If neither 'raspberrypi' nor 'raspberrypi.local' work, try turning your Raspberry Pi off and then back on, and try again.

(l) You should now be able to log in to the Raspberry Pi

(m) The default username for the Raspberry Pi is 'pi', and the password is 'raspberry'.

33. *Setting up the User Interface and Controller.*

(a) To activate the UI and the Controller at the same time, simply navigate to the NYSG directory and run the following command:

```
./start-system.sh
```

(b) To activate just the UI (without the Controller), run the following command:

```
./start-ui.sh
```

Note: Without the Controller running, all functionality will be limited.

34. *Setting up the Jupyter Notebooks.*

(a) You will need to do steps (a) through (f) only once, the first time that you set up the virtual environment, get the dependencies needed and start the Jupyter Notebook. Every time afterwards, you will not need to do steps (a) through (f). You will instead use steps (i) through (l) every time after you finish the initial installation.

———————————— FIRST TIME INSTALLATION ————————————
–

First check your Python installation version. Make sure it is the latest python version, python 3.7.6. You can check this with the command in your terminal or powershell

```
python3 --version
```

It should say:

```
Python 3.7.6
```

Versions of Python3 beyond 3.7.6 might be ok, but some of the dependencies might have to be manually upgraded or more dependencies installed.

(b) Next install Python's package manager pip. Run the command below: In Linux/MacOS:

```
python3 -m pip install --user --upgrade pip
```

In Windows:

```
py -m pip install --upgrade pip
```

(c) Install venvm, the virtual environment in which you can develop and see your code running In Linux/MacOS:

```
python3 -m pip install --user virtualenv
```

In Windows:

```
py -m pip install --user virtualenv
```

(d) Now create your virtual environment: In Linux/MacOS;

```
python3 -m venv env
```

in Windows:

```
py -m venv env
```

You can replace env with any other name you want. For example:

```
nysg_env
```

(e) Start your environment: In Linux/MacOS:

```
source env/bin/activate
```

In Windows:

```
.\env\Scripts\activate
```

Remember to substitute in each of the commands above

```
env
```

with whatever env name you gave to the virtual environment. Every time you want to develop on the notebook, remember to open the environment again.

(f) In your environment, load in the required dependencies.

```
pip3 install -r requirements.txt
```

This installs a long list of dependencies for the virtual environment

(g) To add further dependencies that are Python packages, run

```
pip3 install package_name
```

where

```
package_name
```

is the package you want. For instance, if you want package numpy, run this command:

```
pip3 install numpy
```

You should do this every single time you need to add in a new dependent package that we did not have previously.

(h) To freeze dependencies into the requirements.txt for future use:

```
pip freeze > requirements.txt
```

This will save the dependencies you currently are developing with for other people to use with exactly the same packages and versions.

(i) Below are the steps for normal use of the Jupyter Notebook. You must run these steps below every time to start up the Jupyter Notebook and use it properly.

——————————— NORMAL START UP ———————————

To launch Jupyter Notebook in the virtual environment, run this command:

```
jupyter notebook
```

A browser should open, allowing you to see different files. It will list all files in the current directory you are in. Double Click to Open the file titled Programming Tutorial, which you can work on interactively when it opens in a separate tab in your web browser. Go to that tab.

(j) Below are the steps for normal use of the Jupyter Notebook. You must run these steps below every time to start up the Jupyter Notebook and use it properly.

——————————— NORMAL START UP ———————————

To start using the notebook, follow the commands above. You should be able to start the Notebook and begin learning how to program in Python. The instructions to use the notebook are in the Notebook file you just opened.

As a quick summary: The notebook has cells you can click on. These cells are the rectangular partitions on the webpage. Click the arrow to activate and run the cell. A response is displayed under as the response.

You add new cells: in the upper left corner is a button with a (+) plus sign to add a cell. Go to the cell you want to add under, click on the add button, and a cell will occur underneath.

You can also delete cells. There is again a button in the top bar to delete a cell.

If your notebook somehow crashes, close the window, and restart using the process below.

Don't skip cells! This can cause unintended errors or crashes. Cells evaluate at the order they are run, so skipping cells changes the intended running order. You can imagine what kind of issues this could cause...

If you skip a cell, go back to the cell you skipped and run it, then run the remainder of the cells you skipped ahead on after, in descending order down the page.

(k) Below are the steps for normal use of the Jupyter Notebook. You must run these steps below every time to start up the Jupyter Notebook and use it properly.

——————————— NORMAL START UP ———————————

To close the Jupyter Notebook, close the browser page, then go back to your terminal and run the command ctrl - C to terminate, and enter Y for yes.

(l) Below are the steps for normal use of the Jupyter Notebook. You must run these steps below every time to start up the Jupyter Notebook and use it properly.

——————————— NORMAL START UP ———————————

To leave the virtual environment, run this command in your virtual environment

```
deactivate
```

This returns you back to your terminal. You can now do what you would like in the in the terminal again.

(m) To get the original source for this information, go to this URL: `https://packaging.python.org/guides/installing-using-pip-and-virtual-environments/` This can help you set up and debug the pip3 installation and virtual environments.

(n) Common Issues: You must run

```
pip3
```

and ***NOT***

```
pip
```

Running just pip will cause issues. Remember the 3!.
Also, remember to use python3 as well. You must run

```
python3
```

and ***NOT***

```
python
```

Another issue that comes up recently with installation of scipy, if every requested: If scipy installation causes issues, run this command.

```
python3 -m pip install --user numpy scipy matplotlib ipython jupyter pandas
```

Otherwise, try this:

```
pip3 install scipy==1.0.0rc2
```

And if neither of those work and you are on Windows OS:

```
pip3 install scipy-0.16.1-cp27-none-win_amd64.whl
```

You may also have issues installing numpy. If so, run these commands: Uninstall numpy

```
pip3 uninstall numpy
```

Then install numpy again using a different method.

```
sudo apt-get install python3-numpy
```

Common errors with using the Jupyter Notebook include a notebook that runs forever. If so, you can turn off the notebook engine by clicking on the top bar , clicking

```
kernel
```

and then clicking

```
restart
```

Command Line Prompt

The Command Line is one of many ways you can interact with your computer. Typically, you would click on buttons or applications when using a program (a graphical user interface). Another way that you can run programs is the command line. The command line allows you to directly enter commands to the computer. For example, in command line mode, you type in commands like "ls" (show all folders and files), rather than clicking on a folder icon in a graphical user interface. One way to understand this, as Aaron Wagner has put it, is that command line interfaces are a way for you to "talk" to the computer. You issue commands, and the computer responds to them. In essence, the command line is just an alternate, more wordier, way of controlling and commanding the computer.

Typically, the command line exists on a terminal program. The terminal program takes the input and the command line interface processes your input commands. In the old days, terminals used be large devices, like teletype devices, which connected to even larger computers which would be in a different location. Nowadays, the terminal program is an emulator. The terminal simulates a terminal connection to the core computer system.

You may have seen examples of terminal emulators. On Mac computers, there is a program named Terminal that allows for command line access to the computer. On Windows, you can download PowerShell, which does the equivalent.

The command line interface is part of the shell of the computer. The shell is the outermost most part of the operating system of the computer. If you remember, the operating system is the main program of the computer: it handles scheduling tasks, management of hardware, and input and output services (think clicking a mouse and having typing on the keyboard). The shell is the wrapping around the operating system. It allows the computer user to interact with the operating system. For example, when you run a program from the command line, you are asking the Operating system to schedule and run that program for you.

Command line interfaces do not exist in one flavor. Different command line interfaces use different languages to allow the user to interact with the computer. For example, Mac computers used to require the bash language to talk to the computer. Recently, Mac computer terminal programs changed to the related zsh language. There are tons of other terminal languages.

Command Line Prompt Continued

There are several basic commands you should learn relating to the command line. For example, ”pwd” on a Mac, ”dir” on Windows, gives your current directory location (think what file you are located in). You will see something like ”/...”. For example, if your current directory given by pwd is ”/Users/johnsmith/Desktop/”, then you are in the Users folder of root, and in the johnsmith folder of Users, and in the Desktop folder of johnsmith. Note that ”/” is the root and everything else a subfolder of root.

”ls” lists all the files and folders in the current directory you are in. You will see all the files like Word documents, Python programs and other files and subfolders. ”ls -l” and ”ls -a” are different combinations of arguments you can add to ls; ls -l gives more folder and file information; ls -a gives secret hidden folders (dot files), that the graphical user interface never lets you see. Note that ”-l”, ”-a” or any other argument like

```
"-<Something>"
```

is called a flag, and flags change what a command does. So, adding the ”-l” and ”-a” flags changes the plain vanilla behavior of ”ls” without flags.

”cd” changes directory. You would put the directory you want to change to after. For example, if you are in directory ”/Users/johnsmith/Desktop/” and in Desktop, you have a folder ”homework”, then ”cd homework” will enter the homework directory. If you do ”cd ..”, you go up one directory. ”..” stands for one directory upwards. Running ”cd ..” in ”/Users/johnsmith/Desktop/homework” returns you back to ”/Users/johnsmith/Desktop/”. Finally, ”cd .” does not change your directory, because ”.” stands for the current directory you are in. So ”cd .” when you are in ”/Users/johnsmith/Desktop/homework” leaves you still in ”/Users/johnsmith/Desktop/homework”.

To run a python program, you can type in the command:

```
"python3  <program_name>.py"
```

where ¡program_name¿ is the name of your program.
You may find yourself sometimes in a position where you need to stop a program. First, you can try to stop it with

```
ctrl - C
```

or Control and the C key
If that does not work, run

```
ctrl - D
```

or Control and the D Key.
If that does not work, just close your terminal, using task manager or Force Quit if you have to.

Moreover, you may want to know what processes you have on your computer.

```
ps
```

lists processes and

```
ps -a
```


lists ALL processes, by number, which is the Process ID or PID.
You can imagine a process as its own world, where programs can execute. Thus, a

Command Line Prompt Continued

To kill a process, type into the command line:

```
kill -9 pid
```

where -9 means to kill and and pid is the process id number.
For example, to kill process 1012, we could run:

```
kill -9 1012
```

Continue to kill processes that you want to end in this fashion.

Other programs like C language programs or Java programs can also be run on the command line as well. Another program we think might be helpful to run is git. Git is a version control program; it tracks what changes you make in a directory, just like Google Docs. And like Google Docs, many people and colleagues can work in the same directory, making different changes, and git will track all these changes.The power of git is that people can create branches for different functionality and work independently of each other. Then, when the branches are combined together into one big product, git will merge the branches together, identifying issues where merges will cause issues.
Finally, you might find it tough to type so much on the command line. As a result, the command line has tab completion: the command line will try to fill in what you are typing when you press the tab key on your keyboard. Suppose you have a folder

```
"long_folder_blah_blah"
```

If you want to go to that folder, just type ”cd l”, then tab, and it will autocomplete. Note that tab complete works only when there is no ambiguity. If you had another folder named

```
"longer_folder_blah_blah"
```

tabbing will only complete to ”cd long” and it is up to you to fill in more to remove ambiguity before autocomplete can work. In the above example, you would have to type in either

```
"_"
```

, an underscore, or

```
"e"
```

to distinguish between the two folders after the first four characters ”long”. After distinguishing the next character, tab complete will continue properly when you press tab.

Python

Python is a popular programming language. It is a multifaceted language that can be used for procedural programming, object oriented programming (which you will learn the basics of), functional programming and scripting. Python is distinct from other languages in its non-C influenced syntax, which requires spaces and tabs as part of the syntax. Python also does not need to have a compiled binary to run. Instead, Python is interpreted: in compilation, a machine translates code down to a simpler language of small commands called assembly, whereas interpreted languages like Python compile code down to a smaller instruction set, then execute the language in a separate language with the help of a special program. Finally, Python does not require type declarations. There is no type checking phase before run time; instead, at run time, the compiler will raise errors if there is a type issue.

Python contains constructs that are commonly used in other languages. Native Data types include integers, floats, strings, tuples, lists and dictionaries. Other data types can be custom defined as classes. Moreover, functions and control flow are all present in Python. We have for loops, while loops, if statements, yield statements, and try - except error handling. Function wise, we have the standard functions in other languages. In addition, there are examples of first-class functions, which are functions that can receive other functions as arguments and themselves return functions. Python also has lambda expressions to simulate anonymous functions. Finally, Python can simulate an object oriented language. Python has the ability to declare classes, and create objects. Indeed, everything in Python is a class: your basic integers, strings, objects and even functions are all classes.

Some of the greatest features of Python are the packages that Python is shipped with. These packages include regular expression packages, website viewing packages, packages for making graphical user interfaces and others. In addition, there is a whole collection of packages that other people made, and that you can use. These include numpy and matplotlib, which are used for many mathematical calculations and visualization routines. Other packages further grow Python's capabilities; you can find these on PyPi online, and download them using Python's package manage3, pip. In fact, there is likely one for any major concept you can imagine.

In fact, this project uses some important packages like Django, numpy, matplotlib and more. Django is a package that creates a framework for creating websites, linking the frontend user experience to the backend code. Numpy is a python package primarily meant for large scale computation and thus includes data structures like arrays and functions to operate in the arrays. Finally, matplotlib is a visualization tool that allows you to create custom charts and plots.

Python also comes in different flavors. Python1 was a complete language but was rather basic in some ways. Python2 improved on python1, by adding in closures, a feature for functions, as well as anonymous functions, formatting features and other elements. In this project, we use Python3, the newest version of Python, which again has new features. One day, perhaps Python3 might even have pattern matching.

Python Continued

While Python may seem like an awesome language, it does have some pitfalls. Its lack of type checking at compile time makes it easier to script and more expressive for the programmer. However, this can mean many errors are only detected when the code is run, which can lead to a long amount of debugging and fixing code. Moreover, Python is not always the right tool for every task. For example, if we want an application that runs in a time sensitive manner, and can always be fast, we might want to write our code in C, because C compiles down to instructions and data types that are easier for the computer to execute. If we want code that runs everywhere, we might consider to use Java. When making a online application, Swift would be a good choice for an IOS app for the Apple Store. Writing to a database would require using the language SQL [”Sequel”]. Formally verifying code would lead us to use languages like Coq. And finally, if we wanted to work with artificial intelligence or a logic based approach, we might want to use Prolog. Clearly, a language's worth is partially based on the task it is meant to solve. And for this Greenhouse, because we wanted a piece of software easy to write and test, but not necessarily fast or secure, we chose Python.

Git

Git is a program meant for version control when developing software. Just like google docs, Git tracks changes when you make changes in a file. Also, similar to Google Docs, Git will allow multiple users to work on the same file. The difference in Git is that users work on their own ”version” or world of the files they are assigned; this is their ”branch”. When it comes time to combine all the work together into one integrated product, each branch is merged one at a time back to the central ”master” branch, which contains the final product.

Along the way, Git will raise issues if a merge causes discrepancies. Suppose that both John and Mary change line 3 of a file, with each person changing it in different ways. When both John and Mary try to merge their files to the master, there is a discrepancy. Whose changes are correct? Either could be valid. Git will not try to choose, but will flag the error. Instead, another individual can choose which changes are valid and commit those changes to the master branch.

Git has many different commands and tools. The most important that you may use are git init, which creates a git repository. A repository is a centralized location to hold files and documents, much like a folder on google drive. Git add adds items that were changed to a staging area. Git commit puts a timestamp and name to the changes you made to the items in the staging area; by committing, you are have added a series of changes to your final product. Git clone allows you to copy someone else's repository onto your computer. And Git pull will pull changes from a centralized repository to your computer, allowing you to see what others have done. Finally, Git push sends your changes from your computer back to the centralized repository, so everyone can see what you have accomplished.

So what are the commands in Git? Well, we can create a git repository as a workspace for development. We do this in the command line with the command:

```
git init
```

We can then make changes in our workspace repository.

To add changes we can do

```
git add .
```

Which adds all files which were changed or created. To add specific files only, replace the dot with the file name of your choice.

To commit changes to our workspace officially, so that we have a way to officially track changes, we can do

```
git commit -m "your message here"
```

Where your message here is replaced by your description of the changes you made. You need the double quotes around the message.

How can we get others changes? We can pull. To do this, we run

```
git pull
```

This integrates changes directly into your repository workspace.

Git Continued

A safer method to avoid integrating changes directly is to run these two commands:

```
git fetch

git merge
```

Where the first fetch command gets changes and the second merge command adds in changes.

Note that merge can be used elsewhere as well. One case is when we have two branches, where a branch is essentially a workstream. To create a branch, run:

```
git checkout -b branch-name
```

where branch-name is your name for the branch. To go to another branch from the one you are on:

```
git checkout branch-name
```

where branch-name is the branch you want to go to. To check the current branches on your computer, run:

```
git branch
```

And to merge changes from branch2 onto branch1:

```
git checkout branch1

git merge branch2
```

Now we can have conflicting issues; what if my work actually ends up deleting or modifying your work? git merge will list out all conflicts with the following symbols:

```
>>>>> master
my code
=======
your code
<<<<< incoming
```

which you have to choose which code to keep. You can keep my code, your code, or a combination of both. It's up to you. Then, you need to commit changes, of course. Remember to commit the changes always! To do this, run these commands:

```
git add .

git commit -m "merge message"
```

where merge message is the message you want and you need double quotes.

To share you code with others, use the push command. Run this command:

```
git push
```

You may need to enter your password and username to git.

How can you share items with git? We can use GitHub, a repository storage service. When we push, we can push onto GitHub and other people, our collaborators, can see our changes. They can they pull our changes. You can imagine GitHub as a form of Google Docs, except mostly for code, and where changes can easily be noted.

Web Components

There are different web components used for the user interface. We wrote our code using Django, which serves as a framework for integrating databases, webpages and backend Python code. Django allows us to easily add in different components to our user interface and backend Python code easily, while also giving us ease of testing and interaction with the website. In essence, it helps mediate the differences between the website and backend.

On the front end, we use three different technologies to help create an interactive experience for the user. First, we use a formatting language named HTML to mark the sections and words we put on the document. When using HTML, we can specify different portions of a webpage, such as different containers, in elements called divs. Inside these divs, we can have paragraphs of text and headers and titles. Additionally, in HTML scripts, we can embed portions of computer language code, which can then be executed when needed. This code is typically JavaScript. Next, we then use a styling language, CSS, to give our sections and words on the webpage a personalized style. CSS can set the character font, coloring, the padding size, the border size, whether a section shows up or not and other features. You can imagine CSS as a way to prettify text on a "boring" document. Finally, we use another language, JavaScript, in order to make the front end web page more interactive. For example, JavaScript can be combined with mouse clicks, swipes, keyboard presses and other user actions and responses, and give a response back to the user. A simple example is adding a button, and when the user clicks on the button, JavaScript makes a message pop up. JavaScript allows web pages to become more dynamic, by immersing the user in a deeper experience.

We also concern ourselves with a backend, because we have data that is not stored on the webpage, and must be brought there. On the back end, we use Python. Much of the Python is used to create a logistical system to transfer data back and forth from the user web page to the JSON files in the backend. In addition, we can process the data and send it to different places, with the full capabilities of Python. Note that normally, if we do not use JSON Files, we would use a database system to store data, and for Django and Python, we can interact with SqLite, for which there are provided methods to interface with the given database.

Finally, Django is the middle man between the front and back end experiences. Django funnels information back and forth. Django also provides services to preprocess and render the webpages. Django has some security features to protect against attacks in the web page. Finally, Django also has services on the backend to create apps that interact with each other and with database systems.

Raspberry Pi

The Raspberry Pi is a small and cheap computer, primarily meant for education objectives. It includes a processor and has many input-output ports. Besides the USB ports, HDMI visual port and the charging port, the Raspberry Pi also includes a Mini SD Card holder, which holds the memory for the raspberry pi, as well as the Operating System. In addition, there are 40 GPIO Pins, which allow for control of external devices, such as sensors or peripherals. The Pi also includes wireless capabilities, which allows it to connect to the internet. Overall, the Raspberry Pi is a fully functioning computer.

Machine Learning

Machine learning is an area of study in which computers are taught via with large amounts data to influence their decision making. Machine Learning leverages the fact that computers can calculate very quickly to overcome the issue that computers are daily rigid in what they can do. Thus, we computers can read over and calculate values from data very quickly in order to help us make our decisions.

A concrete example of machine learning can be seen in image classification. Suppose we, a human, have never seen a dog before. We can look at 100 pictures of dogs, and later on, we can infer what a dog is. Essentially, we have a schema of what a dog is. For a computer, building that schema is more tricky. For one, computers have no concept of what an idea is. They can only see data and organize that data. Secondly, computers are rigid in their processing of data. In order for a computer to recognize an image of a dog, it must then be taught to look at the different pixels and recognize similarities between the pixels. To recognize these similarities accurately, we might have to feed the computer hundreds of thousands of images. While this is vastly more time consuming than teaching a human what a dog is, since computers can do calculations very quickly, the time is not as long as you might think.

In general, there are many techniques for machine learning. We can try to create computerized versions of "brain neurons" in neural networks. These neural networks have been trained to do tasks like image and word recognition, car driving and processing of human language and speaking. We could also use tree based methods, support vector machines and other types of methods in order to train out computer.

Today, there are many tasks for which we can use machine learning in. Computer vision, such as self-driving cars, is a major task. Another is computers understanding human grammar and speech. This is the focus of natural language processing. Finally, another interesting task is reading your handwriting, for example, parsing your address. Machine learning has evolved so that the Post Office can now have computers read your handwriting, and sort your mail that way, instead of having a human mail sorter.

JSON

JSON is a data transmission format. Unlike other transmission formats, it is meant to be readable by other humans. Thus, JSON is typically written in a wordy form.

The JSON looks like a dictionary, which maps a key to a value. One JSON file could be:

```
{"name" : Steve,
 "age" : 40,
 "date-of-birth" : "01-02-2003"}
```

Notice how easy it is to tell what each component is. We can also put lists in JSON. For example:

```
{"name" : Steve,
 "age" : 40,
 "date-of-birth" : "01-02-2003",
 "friends" : ["Bob", "Anna", "Mark"]}
```

Finally, we can nest JSON dictionaries inside one another.

```
{"name" : Steve,
 "age" : 40,
 "date-of-birth" : "01-02-2003",
 "friends" : ["Bob", "Anna", "Mark"],
 "social_security" :
        {"benefits" : true,
         "number" : 1010200, "is_government_employee" : false
        }
}
```

In general, JSON can hold string keys, and the values themselves can be dictionaries, lists, strings, ints, floats or booleans. To find the official JSON format, you can search online. Note that JSON keys must be strings, and cannot be integers or booleans or something else like a pair or tuple. Instead, if you have a bunch of information, you could hash it to create a string, combine it, like in Python's

```
datetime.datetime.strftime
```

function, or use another method to create a unique string key.

Python has the package

```
json
```

to read json files. Of course, json, which is named for JavaScript, can be read by JavaScript. And in general, most languages have utilities to read from JSON files.

In this project, we use JSON files to hold information. For example, we can hold your plant profile information. We can also log sensor, and peripheral data, and put alerts in JSON files. Finally, for your email settings, we also put these in JSON files. In general, we just read in a JSON file to get prior information, and write back into the same file to store information for future accesses and uses. In order to coordinate between modules, we agreed on a common format to write to the JSON files and from there adhered to those conventions.

4.2 Soil Moisture Sensor Calibration

The soil moisture sensor will need to be calibrated for the system to work properly. Calibrating it ensures that even if different sensors have different outputs, they can all be scaled to fit.

1. Open terminal.
2. Navigate to the directory NYSG/Controller.
3. Run the soil calibration function by typing

```
python3 soil_calibration.py
```

4. Follow the prompts provided in the function.
5. Open another terminal window and navigate to the same directory.
6. Type

```
nano sensor_class.py
```

7. Find the variables “DRY_SOIL” and “WET_SOIL”.
8. Replace the values with those provided by the soil calibration function.
9. Save the changes by hitting ctrl-x then y.

A. Appendix

A.1 Bill of Materials

Table A.1

Name	#	**Identifier**
Storage Bin	1	Sterilite 26 Gallon Clear
Assorted Zip Ties	15	4”, 6”, 8”
Serving tray top/plastic wrap (optional)		
Old pillowcase, fabric	1	
Drip tray or garbage bag	1	See section on Soil
Temperature / Humidity Sensor	1	SparkFun SEN-13763
Soil Moisture Sensor	1	DFRobot SEN0193
Light Sensor	1	Adafruit 4162
Solenoid Valve	1	Electric Solenoid Valves RSC-1-12VDC
Soaker Hose	3’	1/4”
2L Soda bottle	1	
Vinyl tubing	1’	1/2” ID
Adapter A	1	3/4” GHT Female to 1/2” Barbed
Adapter B	1	1/2” Barbed to 1/8” NPT Male
Adapter C	1	1/8” NPT Male to 1/4” Soaker hose Barbed
Adapter D	1	1/4” Soaker hose barbed
Plant grow light strip	3’	10mm LED strip, red and blue
LED strip to wire connectors	1	10mm LED to wire connector
Fan	1	CUI CFM-4010-13-22
Heater	1	Zerodis 700890368746
GPIO breadboard extension kit OR PCB	1	Frienda B086BSFQKQ
Jumper cables (breadboard only)		
40 pin ribbon cable (PCB only)	1	
Diode	1	1N5819-T
Transistor	4	ZTX651
100uF Capacitor (breadboard)	1	35ZLH100MEFC6.3X11
100uF Capacitor (PCB)	1	EEE-HBH101UAP
1uF Capacitor (breadboard)	2	C333C105M5U5TA7301
1uF Capacitor (PCB)	2	C1206C105Z4VACTU
ADC	1	MCP3001 - I/P
8 position DIP socket (PCB)	1	A 08-LC-TT
1k resistor	4	CFR-25JB-52-1K
100k resistor	5	CFR-25JB-52-100K
Power jack	1	CUI PJ-037AH
40 pin header (PCB)	1	Molex 0010897402
12V Power supply	1	QFWB-30-12-US01
9 position terminal block (PCB)	1	TE 282837-9
12 position terminal block (PCB)	1	Phoenix Contact 1990106

Raspberry Pi 4 B	1	
3-pack Raspberry Pi heat sinks	1	
Raspberry Pi charger	1	5V 3.1A USB-C
MicroSD	1	Transcend TS8GUSDU1
Wire (PCB)	1'	3 conductor solid 22 AWG
Wire	7	2 conductor solid 22 AWG
Wire	7	4 conductor solid 22 AWG
Butt Splices/Dolphin Connectors	21	22AWG

Item	**Backup**
Storage Bin	See Section 2.1
Temperature / Humidity Sensor	MIKROE-3272, Adafruit 3251
Soil Moisture Sensor	Any 3.3V analog sensor
Light Sensor	MIKROE-3444
Fan	12VDC brushless fan, 5V PWM control
ADC	NONE

A.2 Tools

Table A.2

Tool	Additional Components	Specs
Drill	Drill bits	Bit sizes: 1/8”, 3/8”, 1/2”, 5/8”,1-1/2”
Utility knife		
Scissors		
Tape measure	Ruler	
Wrench		Crescent, pipe
3D Printer		47mm x 36mm x 36mm
Die (threading)		1/8”-27 NPT
Soldering iron	Solder Wick Flux	
Tweezers		Anti-static
Wire cutters	Wire stripper	
Pliers		Needle-nose Crimping
Screwdrivers		Precision, Flat head and Phillips
Tape		Electrical, Teflon
Multimeter	Probes	

A.3 Acronyms

Table A.3

Acronym	**Meaning**
AC	Alternating current
ADC	Analog-to-digital converter
ANA	Analog
AWG	American wire gauge
BJT	Bipolar junction transistor
CE0	Chip enable 0
CFM	Cubic feet per minute
CLK	Clock
CSS	Cascading Style Sheets
CTRL	Control
DAC	Digital-to-analog converter
DC	Direct current
GHT	Garden Hose Thread
GND	Ground
GPIO	General Purpose Input/Output
HTML	Hypertext Markup Language
I2C	Inter-Integrated Circuit
IP	Internet Protocol
JS	JavaScript
JSON	JavaScript Object Notation
JST	Japan Solderless Terminal
LAN	Local area network
LED	Light-emitting diode
MISO	Master In Slave Out
ML	Machine learning
MOSFET	Metal Oxide Silicon Field Effect Transistor
NPT	American National Standard Pipe Thread
NPN	Negative-positive-negative BJT
NYSG	New York Smart Greenhouse
OS	Operating system
PCB	Printed circuit board
PTC	Positive temperature coefficient
PWM	Pulse Width Modulation
RAM	Random access memory
RL	Reinforcement learning
SCL	Serial Clock
SDA	Serial Data
SPI	Serial Peripheral Interface
SSH	Secure Shell
SSID	Service Set Identifier
UI	User Interface

B. User Interface Documentation

B.1 Overview

The purpose of this user interface (UI) is to provide functionality for remote user interaction with the Smart Greenhouse system. The UI will consist of five pages: Dashboard, Analysis, Germination, Settings, and Information. The UI will allow the user to select healthy levels for temperature, humidity, soil moisture, and sunlight (collectively referred to as "healthy levels"), set the operational mode for the system, and view visualizations related to the system status.

B.2 Usage

The User Interface consists of 6 pages (Dashboard, Analysis, Germination, Information, Weather, and Settings). Lower-level details are explained in the sections below, but high-level notes will be included here. The Dashboard page is intended to provide the user with a simple and easy-to-understand interface that will allow them to monitor the system's performance. The Analysis page is intended to give the user a more in-depth look at the system's behavior over time. The Germination page is intended to allow the user to track a seedling's progress towards germination. The Information page is intended to give the user an overview of care advice for specific types of plants. The Weather page is intended to give the user an overview of current and future weather conditions. The Settings page is intended to give the user a common interface to adjust system-level settings.

B.3 Dashboard Page

The Dashboard page receives bucket numbers and labels for the current healthy levels stored in healthy_levels.json, as well as the sensor data for the updates contained in log.json.

B.3.1 Current Levels

The Current Levels plot compares the current healthy level buckets to sensor readings in the last system update. In Machine Learning mode (see below), the value buckets for temperature, humidity, and soil moisture are represented as light green horizontal ranges, and the current readings are represented as dark green vertical lines. When the system is performing optimally, the dark green lines should be in the center of the horizontal ranges, indicating that each environment variable is reading at the middle of its healthy value bucket. Functionally, this is achieved by loading the page with all ranges and values set to 0 (all horizontal ranges and vertical lines set to the left-most position). Note that the LED light is not controlled via the ML algorithm, so there is no value bucket range for that parameter. In reality, the current levels will likely oscillate about the center of these ranges without necessarily staying exactly centered. This is because the system is subject to significant influence by the external environment (hence, the point of the machine learning).

B.3.2 Historical Levels

The Historical Levels Plot utilizes the log data to compare sensor readings over time. These values are extracted in JavaScript and displayed as a simple line plot using the chart.js plugin which is automatically loaded on each page load. Each environment variable is plotted as a separate line on a value vs. time plot.

Scrolling is allowed to look through all the data, scrolling past and into the present, You can also jump to the most recent points in time. Finally, you can zoom in and out of the plot to see all the data or to zoom into specific parts of the data. the zoom and scroll functions were done client side dynamically with JavaScript, while examines the data set and modifies what is plotted in real time.

B.4 Analysis Page

The Analysis page receives historical performance data from log.json. It displays one large plot containing two data series: sensor data and action data. Sensor data is represented by the green line datapoints. Action data is represented by the blue bar datapoints. All backend data fetching is performed in scripts/data_handler.py, and all frontend processing and display is performed in a JS script in analysis.html. The plot itself is created using the Chart.js plugin, configured in the JS script mentioned above.

Scrolling is allowed to look through all the data, scrolling past and into the present, You can also jump to the most recent points in time. Finally, you can zoom in and out of the plot to see all the data or to zoom into specific parts of the data. the zoom and scroll functions were done client side dynamically with JavaScript, while examines the data set and modifies what is plotted in real time.

There is also a table of statistics that the user might find useful. These include statistics for quantiles, means, medians and ranges for each different variable, like light level, or fan action. These are dynamically generated on the webpage from the Django-python backend and loaded onto the page each time the page is reloaded. The data os this always accurate. Notably, the statistics display the averages of the psyt 10, 20 and 30 data points, as well as all time data. All the data comes from the views file from which data is pulled using a datahandler from JSONS. the views file then cleans the data calculates the values needed and sends it to the rendering function, which creates the final HTML function.

B.5 Germination Page

The Germination page receives information from germination.json. It displays a pie chart of the total time elapsed since planting vs. total time until germination. Backend data fetching is performed by scripts/data_handler.py, and all frontend code is contained in templates/Analysis/germination.html. It also shows start date, current date, end date, days elapsed and days remaining in the germination period.

The start and end dates for germination are configurable int eh advanced settings of the settings page. these influenced the elapsed time since planting and total time until germination.

B.6 Settings Page

The Settings page receives a dictionary of healthy levels, a plant profile string, the healthy levels form, and the plant profile form. The healthy levels dictionary is the return of data_handler.read_healthy_levels(), which contains the healthy levels data from healthy_levels.json. The plant profile string is the return of data_handler.read_plant_profile(), which contains the name of the current plant profile

stored in plant_profile.json. All backend data fetching is performed by scripts/data_handler.py. Frontend code is distributed between the Settings directory and templates/Settings/settings.html. All backend data is stored in the Interface Files directory.

B.6.1 Modes

Machine Learning In Machine Learning mode, the user can specify healthy levels for a plant. In Machine Learning mode, the machine learning algorithm (Machine Learning/reinforcement-learning_static.py) will be used by the controller (controller_main.py) to choose actions to automatically approach those levels. Note that the algorithm needs time to learn its environment, so you may need to leave the system running for a few hours before the algorithm performs with relative accuracy.

Plant Profile The Plant Profile form allows the user to choose a profile with an associated set of healthy levels. Some profiles are pre-loaded with the system, but by creating a new set of levels using the Healthy Levels form, the user can create a new profile. Creating a new profile will create a new profile instance in the system and will be available for future use. All profiles are stored in Interface Files/healthy_levels_by_profile.json.

Healthy Levels The Healthy Levels form allows the user to view the healthy levels associated with a profile, as well as create new combinations of levels. When a new combination is created, the "Save as New Profile" button will appear. Clicking this will allow the user to create a new profile with the designated healthy levels. Expanding the Sidebar (see below) will allow the user to view the real-world values associated with each healthy level's available values. The user may reference the Information page (see below) for more information on what levels might be associated with a certain type of plant.

Manual In Manual mode, the user can specify actions which the system will take. This mode will not utilize the machine learning algorithm, and will instead take actions based only on these settings.

The user can specify how long each peripheral should be turned on (0, 30 and 60 seconds) and which peripheral (fan, heater, water valve and plant light). In advanced settings, PWM settings like frequency and duty cycles for many different setting levels can be specified for the fan and plant light. Also, the update cycle in seconds (60, 180 and 300 seconds) can be specified for each peripheral update in the advanced settings.

B.6.2 Advanced Settings

The Advanced Settings section can be expanded by clicking the arrow button to the right of the header. This section contains settings which affect the system's lower-level operation. Note that adjusting any performance-related settings will alter the applicability of the Machine Learning algorithm's prior learning, and may result in a period where the algorithm needs to re-learn its environment based on new parameters.

Advanced settings include: setting the temperature settings to be Celsius or Fahrenheit, setting the germination time, based on start and end date, or based on how many days forward from the current date the user wants, setting the update speed, setting the email settings to get detailed or not detailed messages, and how long before an email is sent to the user, the user's personal email and the password (do not use a important email as password is not secured), setting the frequency of the fan and light, setting the duty cycles for the fan and light. The user can enter their address including street address, city, state and zip code to get the weather report customized for their region. Besides these settings, the user can also delete their data, if needed, to clear out their data logs. The user deleting data function will store the data in a backup log for the user.

The advanced setting were again crated using Django forms and stored in the JSON files when changes are detected. The JSON files are used to influence all parts of the system when data is read from the appropriate JSON file. Handlers in data_handlers write in the data as necessary and also read data out of the JSON files as well.

B.7 Information Page

The Information page receives information from ui_plant_scrape.json. The data is fetched in scripts/data_handler.py and displayed using JS in templates/Information/information.html. All other supporting code is inside of the Information directory. This page is intended to be used as a reference when users are creating new plant profiles.

The information page includes detailed information organized about each plant including the height, width, and soil depth, optimal conditions like temperature and water, lifecycle of plant, plant growth difficulty, rain level needed and sun light needed. An extended discussion section includes pests, and other interesting parts of the plant information. The information is stored as JSON files in the backend and fetched and displayed. Moreover, the data was scraped using python with the help of Jupyter Notebooks and other modules like beautiful soup 4 and requests.

On the front end, the page has buttons controlled in real time client side code with JavaScript. This operates what the user can see; clicking on a button opens a card, and reveals more content while closing the button similarly closes the card. there are card information within card as well, which includes the hiding of the detailed information until the user wants to read it.

B.8 Weather Page

The weather page displays real world current weather from the National Weather Service and allows the user to read the information. This includes temperature, precipitation, storm alerts and other data including highs and lows.Wind speed and direction are shown on the page. A 7 day forecast is given as well. The page changes dynamically in the day and night time, and there are different background colorings, while the image also changes based on the forecast. For example, at night, there are night images, in the sunrise time, there is a sunrise image, while at sundown, there is a sundown image. If the temperature is above 90, a warm weather image shows up, and if it snowing, cloudy, raining or there is a storm, the appropriate image of snow, clouds, rain or a storm will show. In addition, National Weather Service mini-weather icons display quick to read information. Your location that you input on the advanced settings influences the forecast; changing your location will change the subsequent forecast that you get. Finally, the date, the time, and your location are featured on the page based on your settings.

The page is dynamic and takes the latest weather data. This updates once a minute to keep pulling data onto the webpage. Further more, if the API for the National Weather Service fails, the exception is caught and the page displays and error message warning the user that the data can not be reached at the present moment.

The temperature is reported in Celsius or Fahrenheit. What is reported depends on your settings in advanced settings.

B.9 Sidebar

The Sidebar element provides the user with a readout of the real-world values for the last sensor update. Also, frequency and duty cycles are displayed for the user from real-time. It also includes a section that displays the mapping between value buckets and the real-world values. All data is fetched via scripts/data_handler.py from Interface Files/log.json and Interface Files/value_buckets.json.

B.10 Starting The UI

The UI is automatically started alongside the controller when running the start-system.sh bash script. To run just the UI (without the controller), the start-ui.sh bash script can also be run.

B.11 Accessing The UI

The UI can be accessed by any device connected to the same LAN as the system by navigating to the web address listed in the return message of the bash script.

C. Machine Learning Documentation

C.1 Overview

The machine learning algorithm written for this project (Machine Learning/reinforcement_learning-static.py) is a reinforcement learning algorithm which attempts to learn the effects of distinct combinations of actions on the interior climate of the greenhouse system and use these learned effects to create an optimal policy which chooses the action most likely to bring the system closest to its goal state at each update. Specifically, the algorithm utilizes dynamic programming techniques with an Agent/Environment paradigm. The actions available to the Agent (which represents the physical output peripherals of the system) are all combinations of water, fan, and heat actions, where each action dimension can take on a single value from the set ['off', 'low', 'high']. The dimensions of the measurable Environment (which represents the physical sensors of the system) consist of temperature, humidity, and soil moisture values. The algorithm seeks to take actions in the (water x fan x heat) action space in order to reach states in the (temperature x humidity x soil moisture) space that approach the goal state as defined by the system.

C.2 Agent

The Agent in this algorithm is an abstraction of the output peripherals of the system. These peripherals consist of the watering system, fan, and heater. At each update, the Agent seeks to achieve the maximum expected reward, as returned by a dynamic programming process (see reward structure and considerations below).

C.3 Environment

The Environment in this algorithm is an abstraction of the climate and input sensors of the system. These sensors consist of the temperature/humidity sensor and the soil moisture sensor.

C.3.1 Transitions

Transitions are based on the data contained in Machine Learning/Files/transition.json. This file contains weighted records of all of the actually-observed transitions of the system. On each update, action sets are mapped to effect sets based on value buckets of the system. This recording process is initialized in controller_main.py, and handled in Machine Learning/transition-static.py. The handler function for this process (putEffect()) records a linear combination of the actual observed transition and baseline assumptions formed during the data collection process.

C.3.2 Rewards

The reward structure is intended to encourage continual movement towards the goal state. It determines the Agent's reward based on three parameters: total distance from the goal state, movement towards or away from the goal state, and the number of readings which are in their healthy level ranges. The initial reward value is equal to (-100 * distance from goal state), where the distance from the goal state is calculated by computing the value of abs(current_state.temperature -

goal_state.temperature) + abs(current_state.humidity - goal_state.humidity) + abs(current_state.soil_moisture - goal_state.soil_moisture). Then, that value is modified based on the relative movement with respect to the last observed state. For each Environment dimension, if the Agent has moved towards the value for that dimension of the goal state, the reward is increased by 100. If the Agent has moved away, then the reward is decreased by 500 (for soil moisture, this penalty is increased to 700 to strongly discourage flooding the system). Finally, for each dimension, if the current state's value is within the value bucket range denoted by the goal state, the reward is increased by 20.

D. Controller Documentation

This is the main controller for the greenhouse. It is responsible for initializing the software required, and running a main loop by reading sensor data, using the ML algorithm to decide on a decision; or using manual control to influence the actions, logging the data and decision, sending any alerts as specified by the user and then responding to the decision by changing the peripherals. The controller than pauses until the next cycle of execution.

D.1 Setup

In order to use the controller and run any associated code, you must setup the environment and dependencies.

D.1.1 Software Runtime Requirements

Please activate the I2C capabilities on the Raspberry Pi 4 Controller. To do so follow these instructions:

(these instructions are from: https://www.raspberrypi-spy.co.uk/2014/11/enabling-the-i2c-interface-on-the-raspberry-pi/#:~:text=Method%201%20%E2%80%93%20Using%20%E2%80%9CRaspi%2Dconfig%E2%

Please also activate the SPI capabilities on the Raspberry Pi Controller. https://www.raspberrypi-spy.co.uk/2014/08/enabling-the-spi-interface-on-the-raspberry-pi/

Method 1: Command Line

1. Open a new terminal tab
2. Run

```
sudo raspi-config
```

3. Select “Interfacing Options”
4. Scroll to the I2C option and activate by selecting
5. Click “” to activate ARM I2C interface
6. Click “” to reboot
7. Then run

```
sudo raspi-config
```

8. Select “Interfacing Options”
9. Scroll to the SPI option and activate by selecting
10. Click “” to activate ARM I2C interface
11. Click “” to reboot

Method 2: User Interface

1. On Desktop go to Menu > Preferences > Raspberry Pi Configuration
2. Select “Interfaces” and set “I2C” to “Enabled”
3. Click “OK”
4. Click “Yes” to reboot
5. On Desktop go to Menu > Preferences > Raspberry Pi Configuration
6. Select “Interfaces” and set “SPI” to “Enabled”
7. Click “OK”
8. Click “Yes” to reboot

Next, you will need to install smbus, spi and I2C tools. To do so, run the following commands on a terminal tab 1. Open a terminal tab 2. Run

```
sudo apt-get update
sudo apt-get install -y python-smbus i2c-tools
```

To install I2C hardware, start by shutting down the Raspberry Pi. Run:

```
sudo halt
```

Wait 10 seconds, disconnect PI power, and connect hardware.

Also run the two commands

```
sudo apt-get install -y python-dev python3-dev
sudo apt-get install -y python-spidev python3-spidev
```

Then run the following commands:

```
cd -
git clone https://github.com/Gadgetoid/py-spidev.git
cd py-spidev
sudo python setup.py install
sudo python3 setup.py install
cd -
```

To check if I2C enabled, power up the Pi. Run:

```
lsmod | grep i2c_
```

If “i2c_bcm2708” is listed, i2c is working.

To check hardware is working: run this command:

```
i2cdetect -y 1
```

You should get a grid of numbers, with the numbers filled in being the i2c addresses. Note that you need a newer Raspberry Pi model for the last command. So i2c should work.

To verify spi: run

```
lsmod | grep spi_
```

and “spibcm2708” or “spibcm2835” should be listed.

D.1.2 I2C clock stretching

1. Open terminal
2. Type

```
sudo nano /boot/config.txt
```

3. Move down to where the line "dtparam=spi=on"
4. After that line, add the line

```
dtparam=i2c_arm_baudrate=10000
```

5. Hit ctr-x and y to save the file.
6. Type "sudo reboot" into terminal and hit enter to restart the Pi

This changes the clock rate for the I2C bus to 10kHz. If this is still too fast, replace 10000 with 5000 and restart.

D.1.3 Software Dependency Requirements

Dependencies Two packages are crucial for Raspberry Pi software for the controller. - RPi.GPIO - gpiozero Several other packages are used jointly in the controller and must also be installed. These include a host of Adafruit custom packages To install, run the commands below.

Installation To install these dependencies, first install pip on your Raspberry Pi. You will need to be on terminal on the command line, type these commands and hit enter after each for them to take effect. If pip3 is installed, run

```
pip3 --version
```

It should show a version number

Also run:

```
python3 --version
```

You'll a version of python3 greater or equal to Python 3.7.6

Next run this command:

```
pip install -r requirements.txt
```

This will install all requisite requirements that the controller requires, including all the necessary packages for the controller. This command will take some time to run since many dependencies must be installed, and started, so be patient.

D.2 Controller

D.2.1 Physical Specifications

- Raspberry Pi 4 B
- 40 GPIO pins
- Power for Raspberry Pi
- WIFI connection on Raspberry Pi 4
- I2C capability
- SPI capability

D.3 Sensor Classes (sensor_class.py)

D.3.1 Description

The sensor class represents a sensor abstraction, that can turn on and off a sensor and read from the sensor. The Sensor object is meant to be compatible with I2C, and GPIOZERO for the SPI capability.

There are also utility functions, summary functions and debugging functions in sensor_class.py

D.3.2 Sensors

There are 3 physical sensors: 1. Light Sensor 2. Temperature & Humidity Sensor 3. Soil Moisture Sensor

There are 3 sensor abstractions (4 classes): 1. Light Sensor 2. Temperature and Humidity Sensor 3. Soil Moisture Sensor

D.3.3 Dependencies

- GPIOZERO, for spi communication, specifically MCP7001 (moisture sensor)
- busio
- board
- Adafruit_PureIO
- adafruit_si7021 (temp humidity sensor_)
- adafruit_veml7700 (light sensor)

D.3.4 Attributes

1. addr - this represents the I2C address for the sensor
2. register - this represents the register holding data relevant to the sensor

D.3.5 Methods

- read: This reads the requested value(s) from the sensor and returns the value
- shut_down : Shuts down the sensor

D.3.6 Classes

There are four classes. Attributes and Methods are documented below

1. Light Sensor
    1. Attributes
        1. channel. - this represents the I2C channel number for the sensor
        2. sensor - this represents the adafruit VEML7700 sensor object
    2. Methods
        1. read_light: This reads the requested ambient light value from the sensor and returns the value
2. TempHumidity Sensor
    1. Attributes
        1. channel. - this represents the I2C channel number for the sensor
        2. sensor - this represents the adafruit si7021 sensor object
    2. Methods
        1. read_temp: This reads the requested temperature value from the sensor and returns the value
        2. read_rh: This reads the requested relative humidity value from the sensor and returns the value
3. Soil Moisture Sensor
    1. Attributes
        1. Sensor - this represents the gpiozero MCP3001 soil moisture object abstraction
    2. Methods
        1. read_moisture: This reads the requested soil moisture value from the sensor and returns the value

D.3.7 Utility Functions

1. Create Channel - this creates a Adafruit bus i2C channel . The channel must be 1.

D.3.8 Summary Functions

1. collect_all_sensors() : this returns all the sensor data from the sensors as a dictionary in the format

```
{str_time: {"temperature" : t, "humidity" : h, "soil_moisture" : m, "sunlight" : s}}
```

where str_time is the current time given by strftime in datetime.datetime up to seconds.

D.3.9 Debugging Functions

1. Run_debug runs a sensor and logs the data for that sensor
2. read_debug_data prints out debug data to screen. Must be run only after a debug function was run first to collect data
3. test_sensor_logging tests sensor logging functionality
4. basic_temp_humid_test makes sure the temperature sensor can be read and prints out the values of temperature
5. basic_light_test makes sure ambient light can be read
6. basic_moisture_test makes sure moisture can be read
7. three_sensor_test makes sure all three sensors can be read at once and prints out temperature, humidity, moisture and light values

D.3.10 Running sensor_class.py

Running

```
python3 sensor_class.py
```

will run the final 4 debugging tests above.

D.4 Peripheral Classes (peripheral_class.py)

D.4.1 Description

The peripheral class represents a peripheral abstraction, that can turn on and off a sensor and changed at the peripheral. The peripheral object is meant to be compatible with GPIO pins.

There are also utility functions and debugging functions in peripheral_class.py

D.4.2 Dependencies

- RPi.GPIO, for GPIO pins communication [Requires a Raspberry Pi for this package]
- Other backup packages
  - gpiozero provides more complex abstractions that may prove fruitful should we need them

D.4.3 Attributes

1. channel - this represents the GPIO pin channel
2. active - this represents whether the pin channel is active or not (on or off)

D.4.4 Methods

- setup: Sets up pin channel and turns on the board settings to BCM at the first peripheral creation
- set_active: makes channel active and makes the GPIO channel respond with high voltage
- set_inactive: makes channel inactive, and makes GPIO channel respond with low voltage
- respond: outputs to GPIO channel with correct voltage level to device to activate or deactivate
- read: reads from GPIO channel
- deactivate: deactivates and closes GPIO channel, cleaning up any remnants of channel and activity

D.4.5 Sublasses

There are two abstract subclasses. They are all subclasses of Peripheral, and have the same base attributes. 1. BurstPeripheral 1. Attributes 1. Burst_time: the time the peripheral runs before turning off 2. Methods 1. set_active sets the peripheral active for burst time seconds and then turns off after 2. Set_inactive turns off peripheral 3. get_burst_time gets burst time in seconds 4. set_burst_time sets burst time for the peripheral in seconds 2. Pwm_Peripheral 1. Attributes 1. freq the frequency in Hertz 2. dc: the duty cycles in percent 3. PWM: the PWM object to control PWM 2. Methods 1. Set_inacvtive: changes duty_cycle to 0 to turn off 2. set_freq sets frequency in Hertz 3. get_freq gets frequency of the PWM device i hertz 4. set_duty_cycle sets duty cycle in % 5. get_duty_cycle gets duty cycle in % 6. deactivate turns off all PWM devices completely

There are 4 concrete base classes: 1. PlantLight 1. Superclass 1. Pwm_peripheral 2. Fan 1. Superclass 1. Pwm_peripjeral 2. Attributes 1. set_duty_cycle sets the duty cycle inverted by subtracting from 100 and taking absolute value 3. HeatPad 1. Superclass: 1. Burst_peripheral 4. SolenoidValve 1. Superclass: 1. Burst_peripheral

D.4.6 Summary Functions

1. react_all - this reacts to all the peripherals, changing them the way requested. Runs asynchronously to allow for yielding
2. translate_action_to_burst_time translates the action to a burst time from interface files
3. change_peripheral changes the peripheral by activating it
4. manual for manual control

D.4.7 Debugging Functions

1. debug_peripheral runs a peripheral and logs the data for that peripheral
2. debug_fan runs debugging for a fan, changes duty cycles and logs the recorded tachometer speed
3. read_debug_data prints out debug data to screen. Must be run only after a debug function was run first to collect data
4. fan_turn_on_test turns on fan
5. test_peripheral_logging tests peripheral logging capability

Running

```
python3 peripheral_class.py
```

will test the fan, check logging and test all four peripherals

D.5 Main (controller_main.py)

Main houses all driver program functions. This is where the function is executed from. Running main will start up the initialization and the event loop to continuously monitor sensor data and also change peripherals. The machine learning algorithm ties into the system in main, as does the manual control.

D.5.1 Dependencies:

pin_constants - these are the constants used sensor_class - sensors imported peripheral class - peripherals imported reinforcement_learning - the ML algorithm

D.5.2 ML Wrapper

1. ml_adapter: the ml function is called from here, and argument dictionaries are used to feed in and pipe out data to this adapter

D.5.3 Initialization

1. init: sets up controller with sensors and peripherals and the associated channels and registers and pins

D.5.4 Event Loop Driver

1. one_cycle_sensors: polls data from all sensors once
2. one_cycle_driver - Testing vehicle for running only one cycle
3. one_cycle_peripherals sends voltages to all peripherals once
4. event_loop: reads in user interface input, measures data from sensor, makes decision, logs data and sends out decision to change peripherals, waits a time amount and restarts cycle
5. ml_adapter calls ml function as an adapter pattern function
6. process_from_ml processes the ml output into a peripheral controllable output
7. process_to_ml transforms the user input into an ML input
8. convert_to_bucket converts regular values into bucket data based on interface file settings from the user
9. main : asynchronous driving function

D.5.5 Asynchronicity

The program is partly asynchronous at the present moment. Notably, async package is used to run some commands in a concurrent fashion, actually passing back and forth function execution when the functions can be idle.

D.5.6 Debugging

async_trial.py is a debugging unit to make sure that asynchronous execution works as intended

D.6 Log (log.py)

Log contains all the logging utilities and functions to help log data in json files

D.6.1 Functions

- get_file_size is the size of the file in bytes
- append_dict adds on to the dict keys that are assumed not to be in the dict
- merge_dict is a utility method that is much slower and adds keys uniquely
- log logs the data, either overwriting prior memory location if the file size is to large or appending data if necessary

D.6.2 Debugging

log_test.py tests the functions in log

D.7 Alert (alert.py)

Alert contains all the alert utilities to monitor, raise and log alerts for the user

D.7.1 Functions

- alert is the collection of alerts raised, which also logs the alert, creates the message and sends the email as necessary with the correct settings for the email. If configuration.py not included, causes error to halt execution, if password/username not correct, catches error and does not send email.
- alert_message_generator creates alerts as needed
- log_alert logs an alert
- low_detail_generator generates low detail alerts (only extreme greenhouse conditions)
- high_detail_generator generates high detail alerts (all greenhouse sensor/peripheral output and extreme conditions
- generate_message creates an email message with subject and body specified by detail waned by user
- send_email sends email to the user at specified email address and password

D.7.2 Classes

Also included is a class: AlertStatus Alert Status is a struct tracking information about alerting and when the previous alert was sent

D.7.3 Debugging

test_email_alert.py test the functions in alert by sending an email to the intended address

D.8 Pin Constants (pin_constants.py)

Pin constants contains pin information like GPIO pins, register number, channel numbers, light constants, alert constants, other path locations and I2C commands.

D.8.1 Functions

Also included are standard json/pickle utility functions used to dump and store data.

D.9 run_greenhouse.sh

DO NOT USE THIS SCRIPT. THERE IS A MODIFIED SCRIPT TO USE

D.10 Set up to Run

To Run this program, run the command in the directory above

```
bash start-system.sh
```

The program will start immediately and will not terminate, unless you type in the keyboard ctrl-c, which cause an exception. In addition, this command will also start the UI so that you can manipulate the program.

E. NYSG Interface Files Documentation

E.1 NYSG Interface Files Documentation

E.1.1 Interface Files

Interface files serve as the exchange and storage medium for information between software subsystems. In this project, we chose to make use of the JSON format (as opposed to .txt or databases) for simplicity and ease of use for the end-users (high school students). These files are also meant to provide configurability to the user. Changes to the information in these files will propagate throughout the entire software system.

The files are read from in both the controller_main.py and also in the settings page, analysis page and dashboard pages of the UI. The settings page also writes back data to the interface files, so that data can be updated. This shows how the data can persist and be updated and changed. Essentially, the interface files are a memory medium, rather like RAM or a database, except we structure it by JSON for simplicity in this small scale project.

E.1.1.1 Healthy Levels (healthy_levels.json)

This file stores healthy levels as specified by the user in the Settings page of the UI. Each level is represented by a bucket number corresponding to bucket in value_buckets.json. These levels serve as goals onto which the machine learning algorithm will seek to optimize.

E.1.1.2 Healthy Levels By Profile (healthy_levels_by_profile.json)

This file maps profiles to sets of healthy levels. These profiles will appear in the drop down menu on the Settings page of the UI as pre-configured plant profiles.

E.1.1.3 Healthy Levels By Profile SAMPLE (healthy_levels_by_profile_SAMPLE.json)

This file maps profiles to sets of healthy levels. These profiles will appear in the drop down menu on the Settings page of the UI as pre-configured plant profiles. **THIS IS A SAMPLE FILE** that allows the user to have a basic idea of what they will see in the user interface and is not meant to be used except as a sample or default method.

E.1.1.4 Log (log.json)

This file holds information pertaining to the updates of the system. It contains one record for each update, labeled with the datetime of that update. Inside of these records, information on the sensor readings is stored, as well as the actions that were taken during that update. All values are presented on a scale from 0-5.9. There are different keys: sunlight, temperature, humidity, soil_moisture, water_action, heat_action, fan_action and light_action.

E.1.1.5 Backup Log (log_backup1.json)

This file holds information pertaining to the updates of the system. It contains one record for each update, labeled with the datetime of that update. Inside of these records, information on

the sensor readings is stored, as well as the actions that were taken during that update. All values are presented on a scale from 0-5.9. There are different keys: sunlight, temperature, humidity, soil_moisture, water_action, heat_action, fan_action and light_action.

This file is a backup file which means when the user clears all data on the advanced settings page, all data is deleted in log.json and exported to log_backup1.json. In the case you clear data again from log, log_backup1 is permanently cleared forever!

E.1.1.6 Sample Log (log_SAMPLE.json)

This file holds information pertaining to the updates of the system. It contains one record for each update, labeled with the datetime of that update. Inside of these records, information on the sensor readings is stored, as well as the actions that were taken during that update. All values are presented on a scale from 0-5.9. There are different keys: sunlight, temperature, humidity, soil_moisture, water_action, heat_action, fan_action and light_action.

This file is a sample file. It is mean t as a sample for the UI for the user to visualize how data would show up and be seen. It is not meant for production use.

E.1.1.7 Manual Actions (manual_actions.json)

This file contains the last-submitted set of manual actions. When the system is in "Manual" mode (as set in the Settings page of the UI), the system will take these actions on each update. These actions are set in the user interface file directly and are reflected in the JSON files.

Actions can be low, high and off, and it is for the water, fan, heater and light peripherals.

E.1.1.8 Mode (mode.json)

This file contains the mode that the system is currently in. The mode is either manual or machine learning. it is read in in the controller_main to decide what happens in the update cycle; either the UI manual actions are taken or the ML actions for that cycle are called and taken. The mode is itself written by the user in the advanced settings tab of the UI.

E.1.1.9 Plant Profile (plant_profile.json)

This file contains the current plant profile (as set in the Settings page of the UI). The profile maps to one of the profiles in healthy_levels_by_profile.json. The profile could be custom, or some other preloaded one like "tomatoes" that the user could not change.

E.1.1.10 Value Buckets (value_buckets.json)

This file maps buckets to nominal values. Buckets are used to interpret data throughout the system. For each environment variable (temperature, humidity, soil moisture, sunlight) there are 5 buckets. Each bucket is associated with a label, and low value, and a high value. The label is used in the UI to associate the bucket with relatively familiar concepts (cold/warn/hot), the low value is the floor of nominal values that would be associated with that bucket, and the high value is the ceiling of nominal values that would be associated with that bucket.

E.1.1.11 ML Training (actions.json)

This file contains all training operations that the ML can take over 64 time steps to check what the ML does and to train it properly. For each time step, there is a decision listed for the water, fan and heater, in terms of high, and low and off status. There are 64 time steps specified in total.

E.1.1.12 Alert (alert_log.json)

This file contains all the alerts that the system alerts by time to the water level that is recorded.

E.1.1.13 Default Home Address (default_home_address.json)

This file contains the default home address in Suitville Maryland for the National Weather Service Address reading. This file **MUST NEVER BE CHANGED** because it is a default setting to display should the user give a faulty or incorrectly entered home address.

E.1.1.14 Email Settings (email_settings.json)

This file contains the user email settings that is in terms of rate of email sending and the detail of the email. This file is changed by the advanced settings of the user page in the UI settings. You can set minutely, hourly and daily updates of emails and high and low settings of detail where high is all information and extreme data while low just shows extreme data. The user changes these settings in the advanced settings.

E.1.1.15 Frequency (freq_settings.json)

This file contains the user frequency settings for the pwm frequency settings for the fan and the plant light. The keys are fan and light and the inner keys is frequency. The user changes the frequency individually for light and fan on the UI advanced settings page. The frequency can be changed in units of 10 from 0 to 500 hertz.

E.1.1.16 Germination (germination.json)

This file contains the germination start and end date with day, month and year. It holds start and end dates. The start **can** be equal to the end date and but it can **never** be after the end date. Note that the UI disallows the start being after the end. Also note that the start can be in the future and the end in the future. The start date could also be in the past. Both start and end are specified in the UI advanced settings where the user manually sets six fields of day, month and year for the start and end dates.

E.1.1.17 Home Address (home_address.json)

This file holds the home address for the user including street address, city, state and zip code. It must be a valid American street address. It could be an invalid address because the UI does not check for invalidity. It is is set in the UI settings advanced settings tab and the user types in street address and city and zip code and state is chosen in a drop down. This influences what weather forecast is received by the user in the weather page. The user should set a local address for local weather forecasts.

E.1.1.18 Constants (interface_constants.py)

This contains constants used for the interface files like log path, germination path and other paths.

E.1.1.19 Update Interval (interval_settings.json)

This file contains the update interval in **seconds**. It could be any value in 60, 180 and 300 seconds. This file can be changed in the User interface advanced settings tab. This file interacts with the controller and tells the controller how long to await for the next update cycle in an await sleep call.

E.1.1.20 ML Log (ml_action_log.json)

This file contains the actions of the ML taken in the last cycle including keys of water, fan heat, light and expected reward. It is logged from controller_main and it cannot be changed or deleted from the UI.

E.1.1.21 Duty Cycles (pwm_settings.json)

This file contains the user duty cycle settings for the pwm duty cycle settings for the fan and the plant light. The keys are fan and light and the inner keys is duty_cycle. The user changes the duty cycle individually for light and fan on the UI advanced settings page. The duty cycle can be cahanged in units of 10 from 0 to 100 % duty cycles.

E.1.1.22 Sensor Log (sensor_log.json)

This file contains the sensor measurements of the controller taken in the all cycles including keys of sunlight, soil_moisture, temperature and humidity with the primary key of a datetime string. It is logged from controller_main and it cannot be changed or deleted from the UI. Sunlight units are unscaled from 0 to 120000 lumens / square meter, temperature is between 0 and 100 degrees F, humidity is between 0 and 100% RH and soil_moisture is inverted and then scaled between 0 and 100 in this log. These are essentially exact readings and not converted to value buckets for the ML!

E.1.1.23 Temperature Settings (temp_format.json)

This file contains the temperature scale in **Fahrenheit or Celsius**. It could be any value in Fahrenheit or Celsius. This file can be changed in the User interface advanced settings tab. Currently, this file only changes what the temperature in the weather page in the UI is displayed as.

E.1.1.24 Utilities (utilities.py)

This contains utilities to translate actions to numbers: e.g. off is 0, low is 2 and high is 4. These functions are called in the controller_main to convert for units for the peripheral changes which only takes in values from 0 to 4.

F. NYSG Controller Construction Guide

F.1 Summary

This guide is meant as a way for you to build your own controller. Follow the instructions carefully and use the Jupyter Notebooks for reference when you build the software controller. You can still use the system without the controller you build. Our controller also works as well, which you can use to compare functionality with.

F.2 Materials

You only need one file: **student_controller_main.py**

DO NOT MAKE EDITS IN ANY OTHER FILES.

For Reference Please reference controller_main.py if you ever get stuck. Also reference the Jupyter Notebook in the folder named Jupyter Notebook.

AGAIN, DO NOT MAKE EDITS IN ANY OTHER FILES.

F.3 The Code You Will Write.

All the code you will write are located in the sections marked at the beginning with ———— EDIT BELOW HERE ————

and ending with ———— EDIT ABOVE HERE ————

DO NOT MAKE EDITS IN ANY OTHER AREAS OF THIS FILE. DO NOT MAKE EDITS IN ANY ABOVE OR BELOW THESE DEMARCATED SECTIONS OF CODE.

There will also be TODO flags inside the code portions where you are to fill in the code and instructions explaining what you need to do.

F.4 Prerequisites and Skills Required

1. Review and Work through Jupyter Notebooks
    1. Learn Assignment and variables
    2. Learn about different data types
        1. strings
        2. lists
        3. ints
        4. floats
        5. booleans
    3. Learn about control flow

    1. if statements
    2. for loops
    3. while loops
  4. Learn Function Calls
  5. Learn how to use external packages and libraries / modules
  6. Learn about classes
  7. Learn how to debug

2. Patience to figure out a solution

3. Ability to reason with code

If you are familiar with programming and with Python, you can consider the prerequisite part 1 to be satisfied.

F.5 Agenda

There are 8 code chunks you need to fill in for the student_controller_main.py file. You can find these by using CONTROL-F or ctrl-F or command-F to search for TODO's, which mark the sections that you are to do complete and fill in.

Here are the descriptions of the tasks.

1. Finish function main() by filling in one line of code. You need to initialize all the sensors and peripherals, which requires calling function init(). Set the result of init to a variable named init_dict, which is a dictionary holding all the sensor and peripherals.

2. Let us create the sensors that we need for this project.

   We need a light sensor, temperature-humidity sensor and moisture sensor. You can create these objects with the constructors: LightSensor, TempHumiditySensor, MoistureSensor, To get more information for each constructor, read the module code in folder Controller and looking in file sensor_class.py

   For the light sensor, use the i2c_channel that was created before this section of code. Similarly, use the i2c_channel for the TempHumidity Sensor. The Moisture Sensor needs to arguments to be called. Save the sensors to variables named light_sensor, temp_humid_sensor and moisture_sensor respectively.

3. Let us create the peripherals for this project.

   We need a Solenoid Valve, Heat Pad, Fan and Plant Light. These are given by the constructors SolenoidValve, HeatPad, Fan, and PlantLight. Each of these takes in a pin number. The pins are: Solenoid Valve: pin_constants.VALVE Heat Pad: pin_constants.HEAT, Fan: pin_constants.VENT, PlantLight: pin_constants.LED

   Further, each of the constructors takes in a burst time. Give them all 20 second burst times. The burst time is an integer.

   The first argument to each constructor is the pin number, while the second is the burst time. There are no other arguments.

   Finally, save each peripheral to its own variable. The variables should be named, respectively, valve, heat, fan and light.

4. We need to be able to read data from the Interface files. We need to read in 5 items: the manual control status, the manual actions, the email settings the pwm settings and the frequency settings.

   TO read in data, call the function load_data. Because load_data is in module pin_constants, you will have to us the module with a dot notation: pin_constants.load_data.

   Load_data takes in one argument, the path of the interface file you want to read. When you can in load_data, load_data returns to you the read in data.

   First, read in manual_control_path and save the results of the function to a variable named manual_control.

   Second, read in manual_actions_path and save the results of the function to a variable named manual_results.

   Next, read in email_settings_path and save the results of the function to a variable named email_settings.

   Then, read in pwm_settings_path and save the results of the function to a variable named pwm_settings.

   Finally, read in freq_settings_path and save the results of the function to a variable named freq_settings.

   Remember, there are five variables you need to create via 5 function calls.

5. We need to convert the healthy light integer into a string. We do it by comparing the value of healthy light.

   If healthy light is greater than or equal to 4, return the string “Full Sun”

   If healthy light is greater or equal to 3 and less than 4, return “Part Sun”

   If healthy light is greater or equal to 2 and less than 3, return “Part Sun”

   If healthy light is greater or equal to 1 and less than 2, return “Part Shade”

   If healthy light is greater or equal to 0 and less than 1, return “Full Sun”

   Remember how to use if statements, elif statements and booleans.

   Do not add in an else clause.

6. We need to convert action to a string. Action can be an integer between 0 and 4 inclusive, or the string “low” or “high”.capitalize

   If action is 0 or it is the string “off”, return the string “off”.

   If action is 1 return the string “big_decrease”.

   If action is 2 or it is the string “low”, return the string “low”.

   If action is 3 return the string “small_increase”.

   If action is 4 or it is the string “high”, return the string “high”.

   Use your boolean comparison equality operation, == , the double equals, as well as if-elif statements. Also return strings inside the if statements.

7. We want you to complete the headers of the while loop. In this branch of the if statement, we want the code to run forever. We can use a while loop with true as the guard. Write in the guard. Then paste in the given code below to call the function to run inside the while loop. You do not need to know what await does. Just paste the code indented inside the while loop body.

   Given code:

```
await one_cycle(init_dict, manual_control_path, manual_actions_path,
   email_settings_path, pwm_settings_path, freq_settings_path,
   sensor_log_path, ml_action_log, alert_log, max_log_size, interval)
```

In this branch of the if statement, we want the code to for a certain number of iterations, max_iter, specifically. max_iter is an integer representing how many iterations to run. We can use a for loop. We want you to fill in the for loop guard. Create an iteration variable. Use range to make sure you iterate for max_iter iterations. Paste in the given code below to call the function to run inside the for loop. You do not need to know what await does. Just paste the code indented inside the for loop body.

Given code:

8. ```
   await one_cycle(init_dict, manual_control_path, manual_actions_path,
      email_settings_path, pwm_settings_path, freq_settings_path,
      sensor_log_path, ml_action_log,alert_log, max_log_size, interval)
   ```

F.6 Testing

To test, run ./start-system.sh

The controller and UI should start. Follow instructions to open the UI and see the results.

F.7 Solutions

Solutions also documented in README-Controller-Main-Solutions.md

F.7.1 Part 1

```
init_dict = init()
```

F.7.2 Part 2

```
light_sensor = LightSensor(i2c_channel)
temp_humid_sensor = TempHumiditySensor(i2c_channel)
moisture_sensor = MoistureSensor()
```

F.7.3 Part 3

```
valve = SolenoidValve(pin_constants.VALVE, 20)
heat = HeatPad(pin_constants.HEAT, 20)
fan = Fan(pin_constants.VENT, 20) \#inverts duty cycles inside class
light = PlantLight(pin_constants.LED, 20)
```

F.7.4 Part 4

```
manual_control = pin_constants.load_data(manual_control_path)

manual_results = pin_constants.load_data(manual_actions_path)

email_settings = pin_constants.load_data(email_settings_path)
```
```

```
pwm_settings = pin_constants.load_data(pwm_settings_path)

freq_settings = pin_constants.load_data(freq_settings_path)
```

F.7.5 Part 5

```
if healthy_light >= 4:
    return "Full sun"
elif healthy_light >= 3:
    return "Part sun"
elif healthy_light >= 2:
    return "Part shade"
elif healthy_light >= 1:
    return "Full shade"
```

F.7.6 Part 6

```
if action == 0 or action == "off":
    return "off"
elif action == 1:
    return "big_decrease"
elif action == 2 or action == "low":
    return "low"
elif action == 3:
    return "small_increase"
elif action == 4 or action == "high":
    return "high"
```

F.7.7 Part 7

```
while True:
    await one_cycle(init_dict, manual_control_path,
    manual_actions_path, email_settings_path, pwm_settings_path,
    freq_settings_path, sensor_log_path, ml_action_log,alert_log,
    max_log_size, interval)
```

F.7.8 Part 8

```
for _ in range(max_iter):
    await one_cycle(init_dict, manual_control_path, manual_actions_path,
    email_settings_path,pwm_settings_path, freq_settings_path, sensor_log_path,
    ml_action_log, alert_log, max_log_size, interval)
```

G. Engineering Explorers Schedule

G.1 Overview and Expectations

We expect students to spend 36 face to face mentor hours on this project, as well as a few hours outside of online meetings. Much of the work can be done in the online meetings; the rest is a small amount of work. Over the course of this project, we expect students to take 9 weeks to build the Greenhouse, averaging just about 4.5 hours per week. This schedule is laid out in such a way to take an average of 4.5 hours per week; ambitious, motivated or exceptionally talented students may feel the need to work faster and can adjust the schedule to finish the project to fit their own needs. In fact, it might be possible to finish the entire project in a week given requisite knowledge of programming and electrical engineering. For most students though, we have created an outline and a timeline for which the students can follow.

G.2 Week 1

G.2.1 Mentorship Activities

- Meet and Greet with your mentor
- Meet other fellow students in the program
- Learn about the Engineering Explorers program and expected goals and outcomes
- Learn about the Greenhouse project, the different components and what you are going to build
- Other ice breaking activities from the Engineering Explorers Program
- Help debug any complications when starting up
- Make sure all supplies accounted for and received by mentees

G.2.2 Software Configurations

1. Follow the Software Installation Guide
2. Flash the Raspberry Pi OS on to an SD Card
3. Load card into the Pi and startup, and setup the PI with setup instructions manual, including setting up mouse, keyboard, HDMI cable to screen
4. Setup Raspberry Pi virtual environment for Jupyter notebooks and download all dependencies and requirements as stated in instructions

G.2.3 Supplies Confirmation

1. Make sure that all hardware materials have been attained. These can be found on the STUDENT HARDWARE SUPPLIES LIST

G.2.4 Hardware Configuration

1. Get acquainted with all the hardware components, specifically the Raspberry Pi Controller, the sensors, the peripherals and the breadboard

G.3 Week 2

G.3.1 Mentorship Activities

- Meet up with your mentors
- Talk about this week's action items
- Start construction and software items
- Fix up any debugging items
- Continue getting acquainted with project
- Answer questions on syntax and semantics of python

G.3.2 Jupyter Notebooks

1. Work on assignments and types in jupyter Notebook, familiarizing yourself with the different types and playing around with each type and using variables
2. Try out different examples on jupyter notebook, including different cells and interactivity. Do exercises.
3. Ask questions if confused. Make sure to understand how assignment works and what types are. Play around with types

G.3.3 Hardware Configuration

1. Start building the Greenhouse, beginning by wiring up circuits following circuit diagram

G.4 Week 3

G.4.1 Mentorship Activities

1. Help open UI
2. Help with student learning UI settings
3. Help install Django or other required dependencies
4. Help with greenhouse connection
5. Talk with mentee about project expectations and how well it is going
6. Review assignment statements and types like lists and strings
7. Explain what JSOn files are, how computers work in terms of operating systems, what memory is and how they interact
8. Explain what git software is and how it is used, explain about command line and clear up difficulties with command line prompt

G.4.2 Jupyter Notebooks

1. Work on control flow functions and modules in Jupyter Notebook
2. Make sure to do exercises on control flow!
3. Review concepts like strings and lists, assignment, how to use strings and lists, how to extract values from lists
4. Understand if-statements, for loops really well
5. Ask questions if confused

G.4.3 Jupyter Notebook

1. Cut fan and input slots
2. Install Raspberry Pi and the breadboard on the greenhouse
3. Add in all sensors and peripherals
4. Ask questions about circuitry
5. Read through text boxes in manual on circuits, and the breadboard components

G.4.4 Software Configuration

1. Play around with UI by getting Django installed
2. Install requirements for uI
3. Open up UI
4. Play with settings
5. Try changing controls in UI
6. Read around on the manual about the OS, main memory and RAM, read other blurbs on Python, raspberry pi, what a JSON file is, and other blurbs like on git and the command line
7. Ask about what a JSON file s
8. Ask about GIT
9. Ask about the command line
10. Play around with git commands
11. Play around with command line commands

G.5 Week 4

G.5.1 Mentorship Activities

1. Talk about jupyter notebook control flow, make sure mentee knows how it works
2. Explain functions, work through making your own function
3. Practice with function exercises

4. Explain how modules work
5. Help checkout hardware peripherals and sensors test
6. Help with debugging
7. Help start UI if needed
8. Answer any other questions with mentees
9. Review types, assignments statements, and control flow

G.5.2 Jupyter Notebook

1. Work on functions and modules in Jupyter Notebook
2. Make sure to understand functions well, and do exercises
3. Do modules exercises and understand how modules are used to separate work into different parts and how something from one module can be imported and used in another module
4. Understand calling functions from another module

G.5.3 Hardware Configuration

1. Check all electronic connections
2. Run sensors and peripherals test
3. Run full system test
4. Make sure all tests work as expected

G.5.4 Software Configuration

1. Continue starting UI again
2. Read more blurbs about software in the guide, try using command line
3. Understand how git was used in this project
4. Start to look at controller main code portions
5. Ask mentor about code portions, ask about confusing ideas and topics, start reading instructions for filling in the code and try to understand what is asked.
6. Start working on portions 1 and 2, advancing to 3 and 4 as needed.

G.6 Week 5

G.6.1 Mentorship Activities

- Debug electronics
- Make sure electronics build safely and correctly
- Clear up an misconceptions with classes
- Explain what a class is

- Explain classes in the project like sensor classes and peripherals
- Explain how real software like Django lets programmers use classes
- Review functions, modules and control flow
- Review types like strings, lists and booleans

G.6.2 Jupyter Notebook

1. Work on classes in Jupyter Notebook
2. Do classes exercises
3. Understand classes well in the context of this project
4. Use new cells in Jupyter to create classes and play around
5. Use the concepts learned earlier like types, assignment, functions and control flow to review material

G.6.3 Hardware Configuration

1. Check all components wired together, like pi and sensors and peripherals
2. Rerun sensors and peripherals test
3. Rerun full system test

G.6.4 Software Configuration

1. Continue to look at Controller Main student page and start to identify code portions and see where they are located
2. Read code portion instructions for 3 and 4 and start to understand task
3. Ask questions or clarify confusion with mentor, and start to fill in next 2 code chunks with mentor's supervision and help
4. Work on code chunks 3 and 4, and move on to 5 and 6 if possible. If too easy, do as much as needed
5. Continue to clarify instructions with mentors and ask questions. Get help to finish.
6. Reread Jupyter Notebooks if needed to get clarification and ask help. Search online if more help is needed.

G.7 Week 6

G.7.1 Mentorship Activities

- Debug code and help with code
- Explain debugging process
- Work through finding and correcting bugs
- Explain print statement debugging
- Explain any confusing concepts like control flow, functions or classes
- Fix up hardware if needed to help student with project

G.7.2 Jupyter Notebook

1. Work on visualization, and debugging in Jupyter Notebook
2. Review functions and classes
3. Add in extra cells to work on it

G.7.3 Software Configuration

1. Start working on tasks 1-8 on the main controller construction guide to build controller
2. Make sure your code matches solution
3. Ask questions to mentor if needed
4. Can check code if done using UI to see if it works

G.8 Week 7

G.8.1 Mentorship Acitivities

1. Help build and debug the main controller that the student will build
2. Give any coding advice as needed
3. Debug any hardware found to be incorrect
4. Weekly meet up and chat with mentors

G.8.2 Software Configuration

1. Continue to Work on tasks 1-8 in the main controller construction guide and build controller
2. Test the controller
3. Make sure hardware peripherals work
4. Make sure data is logged properly
5. Start up UI to see what happens and see if the data is logged
6. Change modes from manual control and machine learning to see if machine learning autonomously controls greenhouse

G.9 Week 8

G.9.1 Mentorship Activities

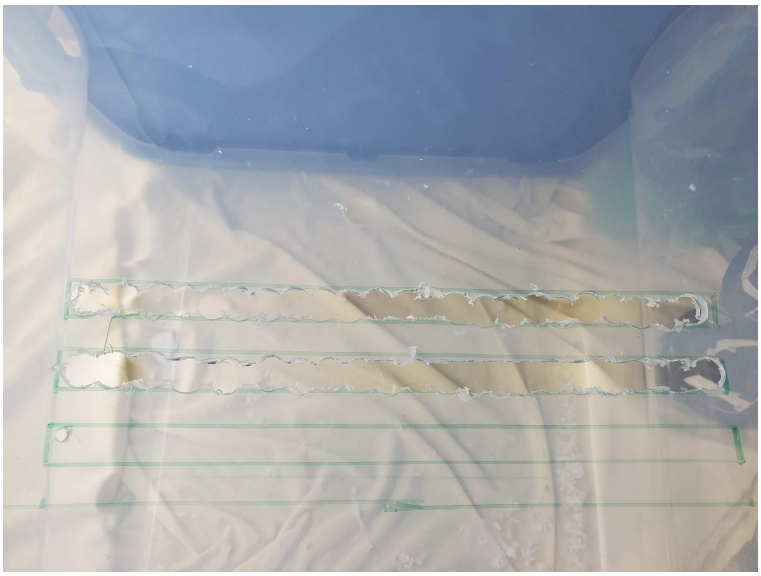
- Wrap up with mentor
- Last meetings and goodbyes

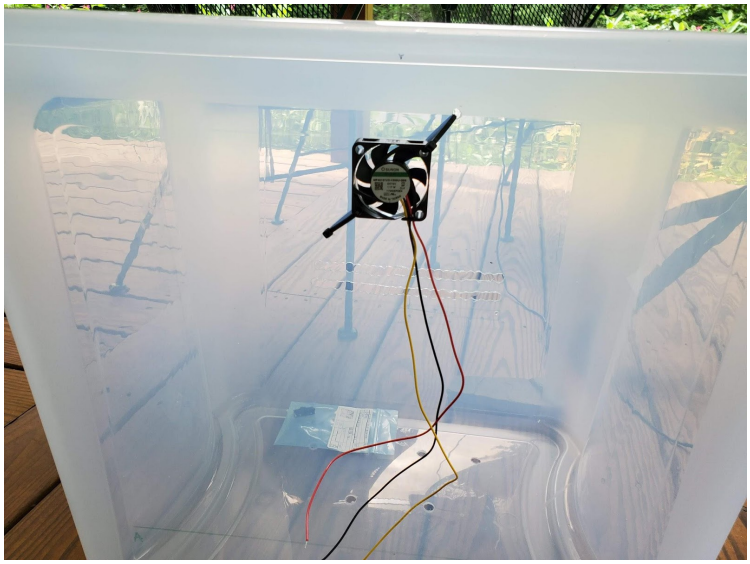
G.9.2 Wrap up

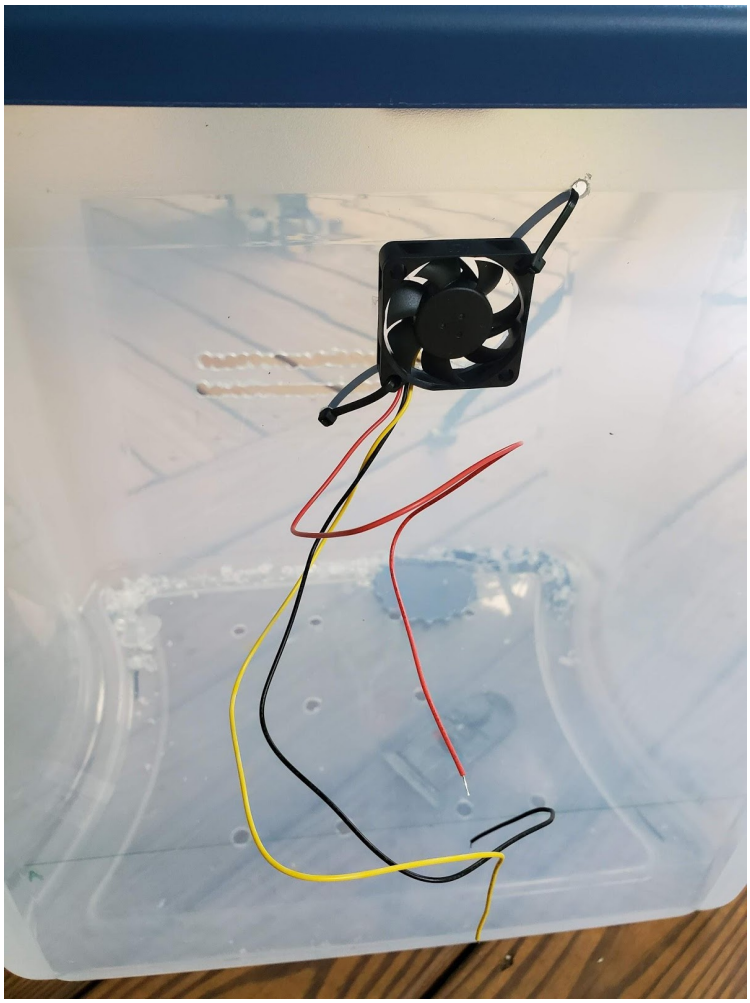
1. Play around and use system with software and hardware
2. Use UI
3. Grow Plants
4. Observe growth!

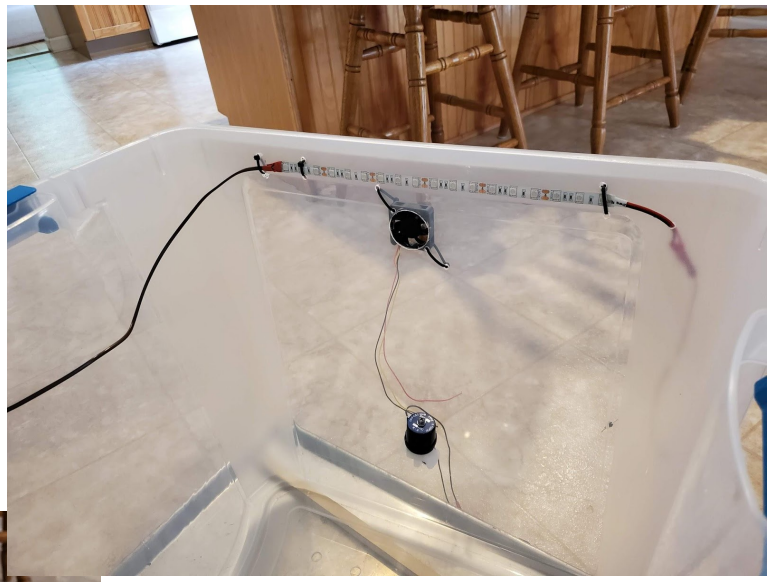
H. Gallery

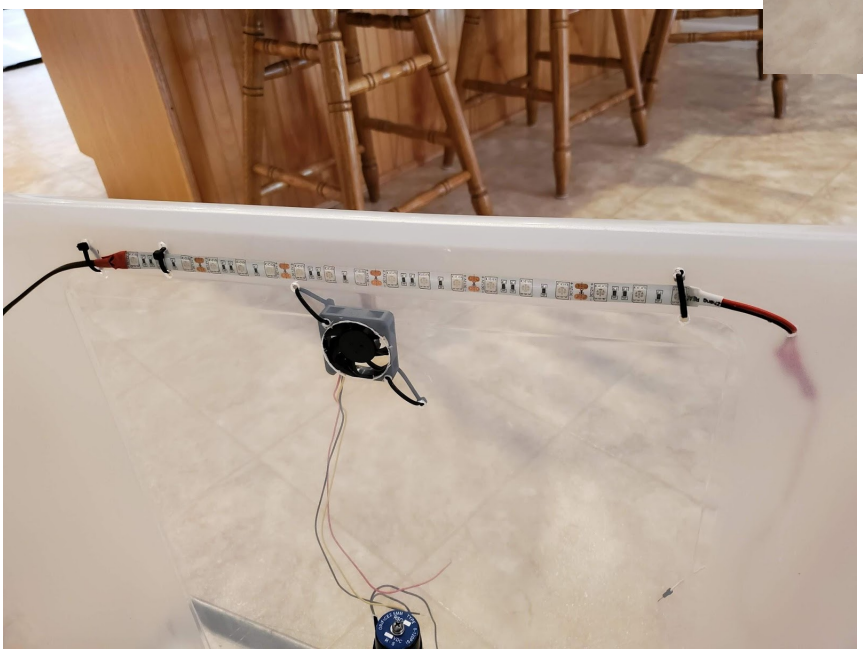
The following pages contain images of the greenhouse throughout its completion. During construction, the design was slightly tweaked. However, they will still provide good reference for most aspects of the project and most design changes will not be visible from the images. Specifically, the fan in some images has the wires in the wrong corner, and the lights were two strips connected by wire instead of a single continuous strip.

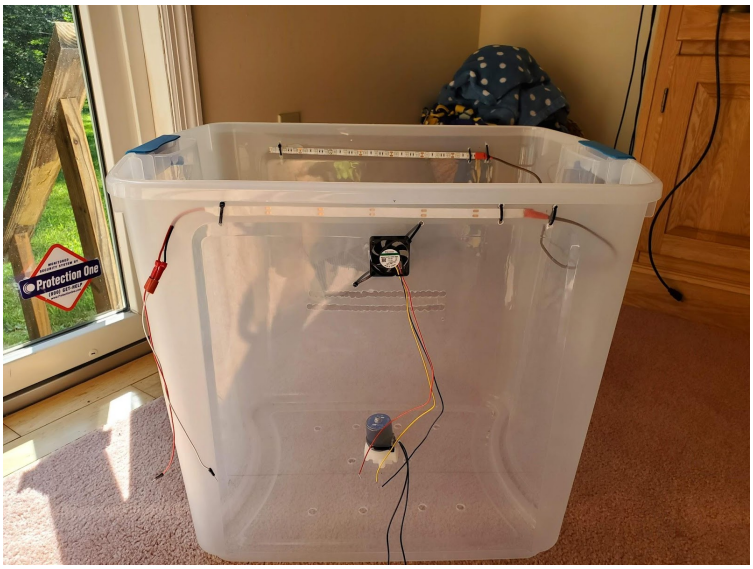


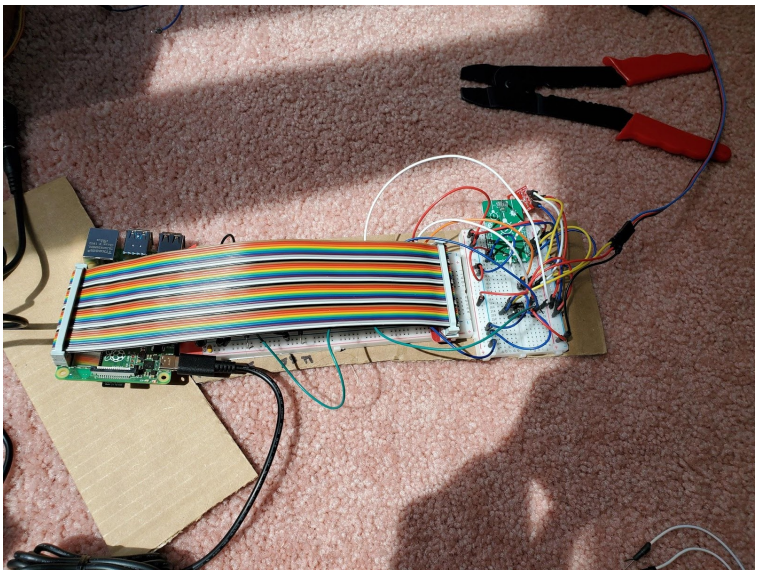


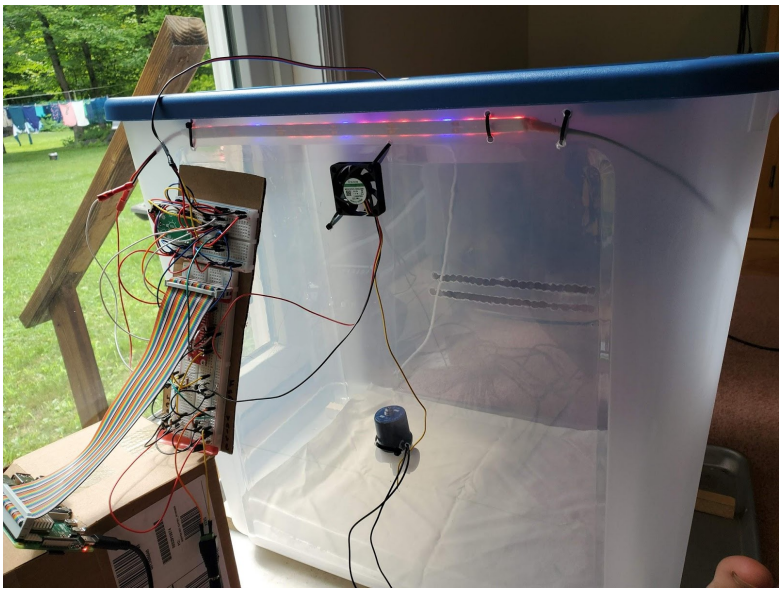




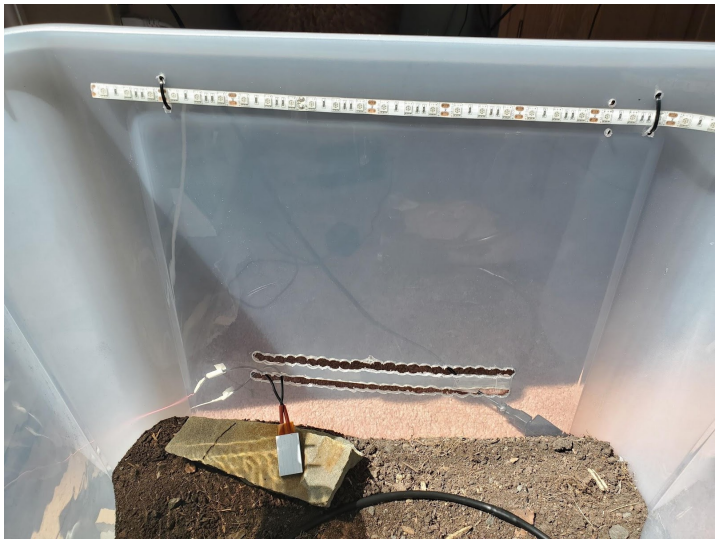


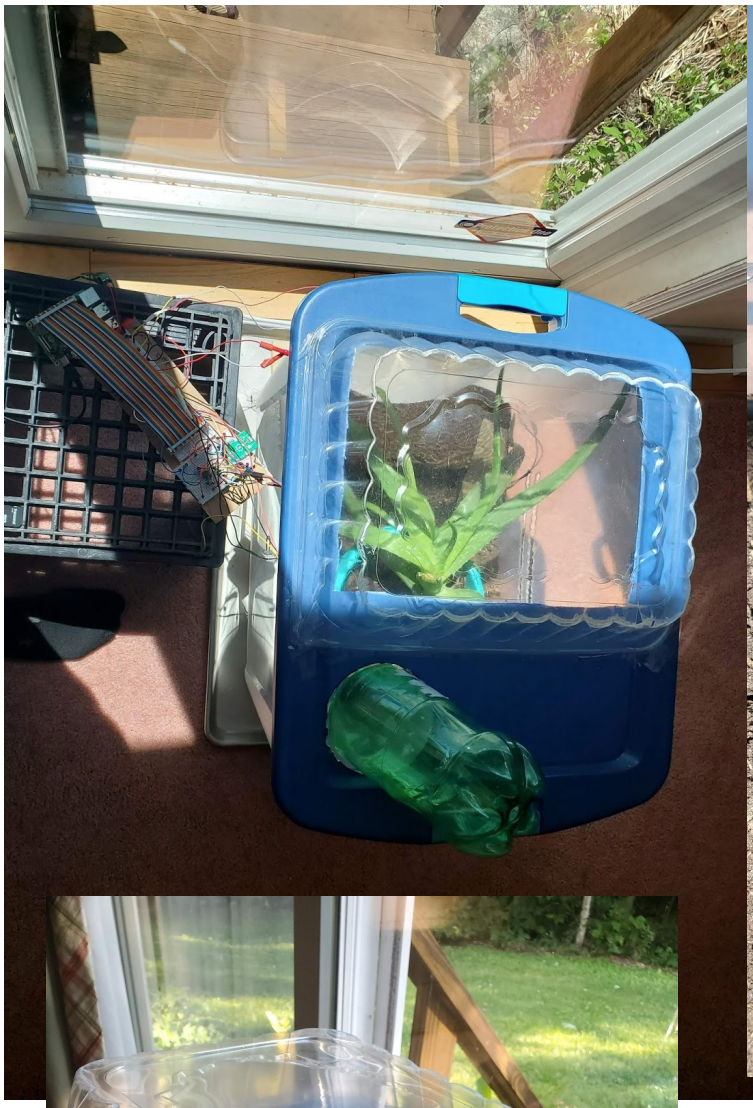

Protection One



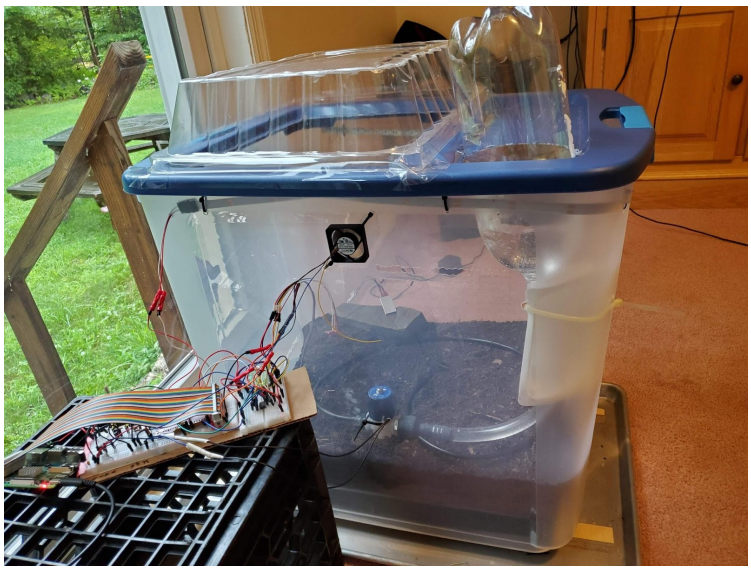


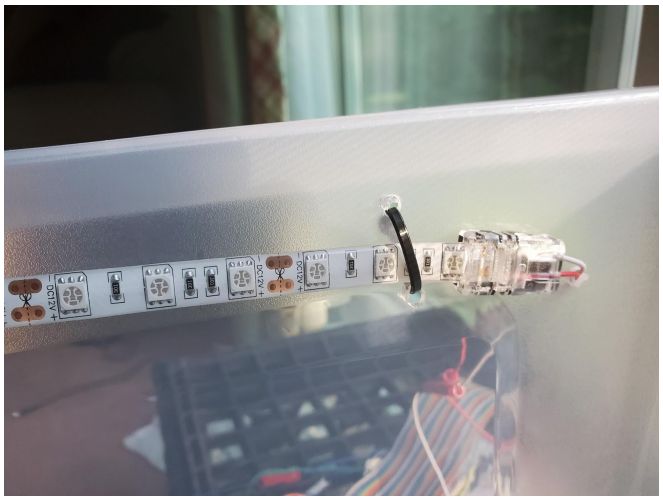

Dean Foods






VDC
Capacitive Soil
Moisture Sensor






AmbientG click

I. Datasheets

The following pages contain the datasheets for important elements of the hardware. In order, they are as follows:

1. NPN BJT (ZTX651) - See I.1
2. ADC (MCP3001) - See I.2
3. Fan (CFM-4010-13-22) - See I.3
4. Light Sensor (VEML7700) - See I.4
5. Temperature/Humidity Sensor (Si7021) - See I.5

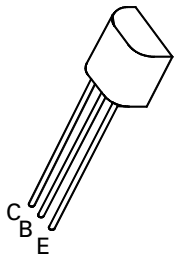
NPN SILICON PLANAR MEDIUM POWER TRANSISTORS

ZTX650 ZTX651

ISSUE 2 – JULY 94

FEATURES

* 60 Volt V_{CEO}
* 2 Amp continuous current
* Low saturation voltage
* P_{tot}=1 Watt



E-Line
TO92 Compatible

ABSOLUTE MAXIMUM RATINGS.

PARAMETER	SYMBOL	ZTX650	ZTX651	UNIT
Collector-Base Voltage	V_{CBO}	60	80	V
Collector-Emitter Voltage	V_{CEO}	45	60	V
Emitter-Base Voltage	V_{EBO}	5		V
Peak Pulse Current	I_{CM}	6		A
Continuous Collector Current	I_C	2		A
Power Dissipation at T_{amb}=25°C derate above 25°C	P_{tot}	1 5.7		W mW/°C
Operating and Storage Temperature Range	T_j:T_{stg}	-55 to +200		°C

ELECTRICAL CHARACTERISTICS (at T_{amb} = 25°C unless otherwise stated).

PARAMETER	SYMBOL	ZTX650			ZTX651			UNIT	CONDITIONS.
		MIN.	TYP.	MAX.	MIN.	TYP.	MAX.		
Collector-Base Breakdown Voltage	$V_{(BR)CBO}$	60			80			V	I_C=100μA
Collector-Emitter Breakdown Voltage	$V_{(BR)CEO}$	45			60			V	I_C=10mA*
Emitter-Base Breakdown Voltage	$V_{(BR)EBO}$	5			5			V	I_E=100μA
Collector Cut-Off Current	I_{CBO}			0.1 10			0.1 10	μA μA μA μA	V_{CB}=45V V_{CB}=60V V_{CB}=45V,T_{amb}=100°C V_{CB}=60V,T_{amb}=100°C
Emitter Cut-Off Current	I_{EBO}			0.1			0.1	μA	V_{EB}=4V
Collector-Emitter Saturation Voltage	$V_{CE(sat)}$		0.12 0.23	0.3 0.5		0.12 0.23	0.3 0.5	V V	I_C=1A, I_B=100mA* I_C=2A, I_B=200mA*
Base-Emitter Saturation Voltage	$V_{BE(sat)}$		0.9	1.25		0.9	1.25	V	I_C=1A, I_B=100mA*
Base-Emitter Turn-On Voltage	$V_{BE(on)}$		0.8	1		0.8	1	V	IC=1A, V_{CE}=2V*

ZTX650
ZTX651

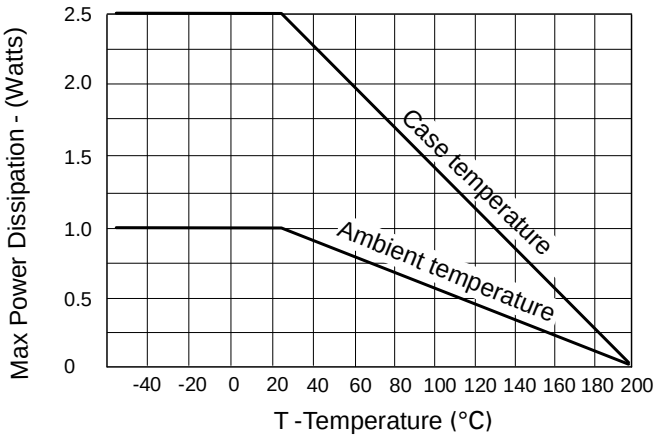
PARAMETER	SYMBOL	ZTX650			ZTX651			UNIT	CONDITIONS.
		MIN.	TYP.	MAX.	MIN.	TYP.	MAX.		
Transition Frequency	f_T	140	175		140	175		MHz	I_C=100mA, V_{CE}=5V f=100MHz
Switching Times	t_{on}		45			45		ns	I_C=500mA, V_{CC}=10V I_{B1}=I_{B2}=50mA
	t_{off}		800			800		ns	
Output Capacitance	C_{obo}			30			30	pF	V_{CB}=10V f=1MHz

*Measured under pulsed conditions. Pulse width=300μs. Duty cycle ≤ 2%

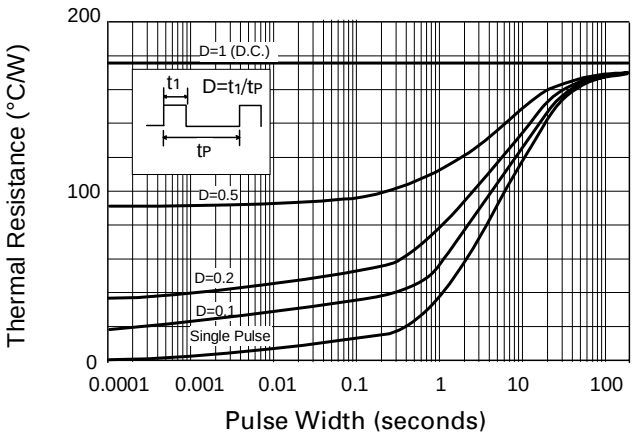
THERMAL CHARACTERISTICS

PARAMETER	SYMBOL	MAX.	UNIT
Thermal Resistance:Junction to Ambient$_1$	$R_{th(j\text{-}amb)1}$	175	°C/W
Junction to Ambient$_2$	$R_{th(j\text{-}amb)2}$†	116	°C/W
Junction to Case	$R_{th(j\text{-}case)}$	70	°C/W

† Device mounted on P.C.B. with copper equal to 1 sq. Inch minimum.

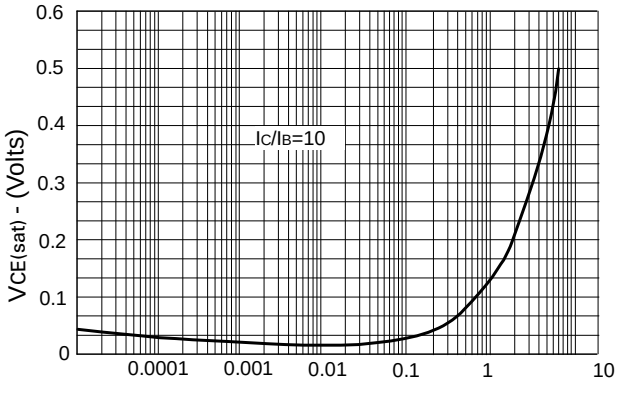


Derating curve

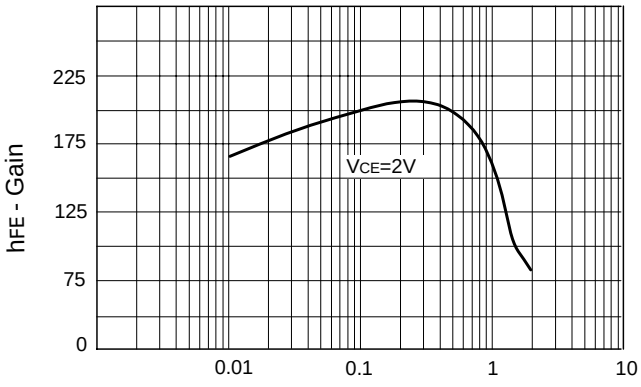


Maximum transient thermal impedance

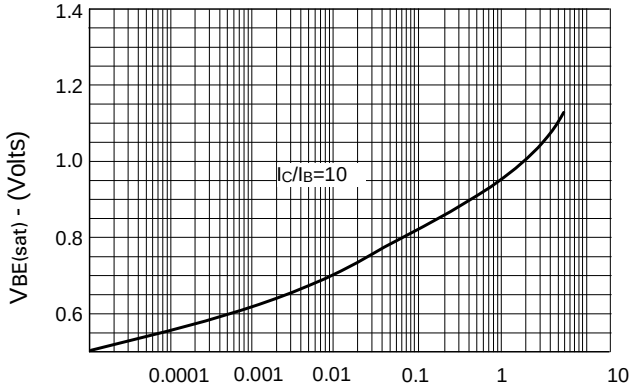
TYPICAL CHARACTERISTICS



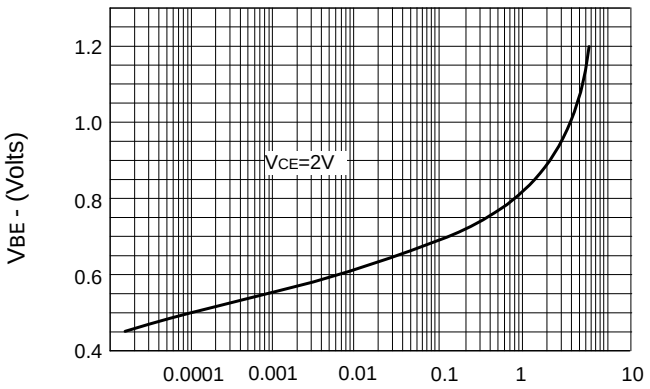
VCE(sat) v IC



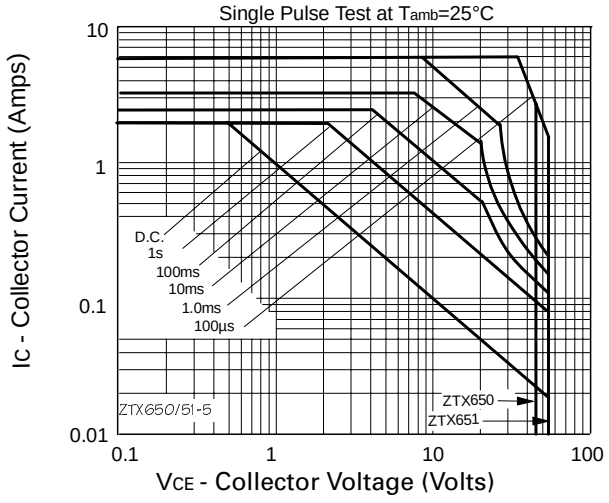
hFE v IC



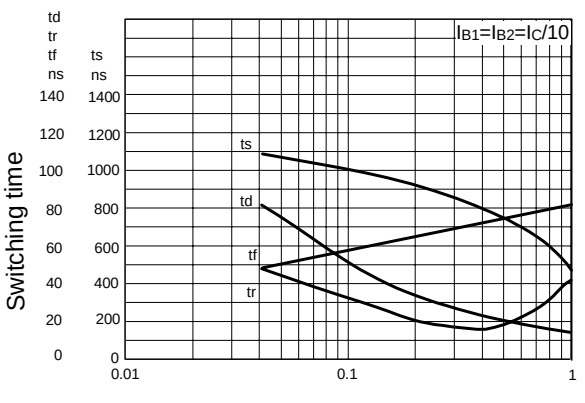
VBE(sat) v IC



VBE(on) v IC



Safe Operating Area



Switching Speeds

MCP3001

2.7V 10-Bit A/D Converter with SPI™ Serial Interface

Features

- 10-bit resolution
- ±1 LSB max DNL
- ±1 LSB max INL
- On-chip sample and hold
- SPI™ serial interface (modes 0,0 and 1,1)
- Single supply operation: 2.7V - 5.5V
- 200 ksps sampling rate at 5V
- 75 ksps sampling rate at 2.7V
- Low power CMOS technology
  - 5 nA typical standby current, 2 µA max
  - 500 µA max active current at 5V
- Industrial temp range: -40°C to +85°C
- 8-pin PDIP, SOIC, MSOP and TSSOP packages

Applications

- Sensor Interface
- Process Control
- Data Acquisition
- Battery Operated Systems

Description

The Microchip Technology Inc. MCP3001 is a successive approximation 10-bit A/D converter (ADC) with onboard sample and hold circuitry. The device provides a single pseudo-differential input. Differential Nonlinearity (DNL) and Integral Nonlinearity (INL) are both specified at ±1 LSB max. Communication with the device is done using a simple serial interface compatible with the SPI protocol. The device is capable of sample rates up to 200 ksps at a clock rate of 2.8 MHz. The MCP3001 operates over a broad voltage range (2.7V - 5.5V). Low current design permits operation with a typical standby current of only 5 nA and a typical active current of 400 µA. The device is offered in 8-pin PDIP, MSOP, TSSOP and 150 mil SOIC packages.

Package Types

PDIP, MSOP, SOIC, TSSOP

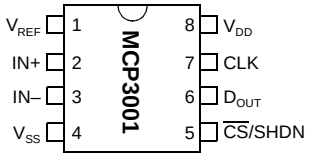
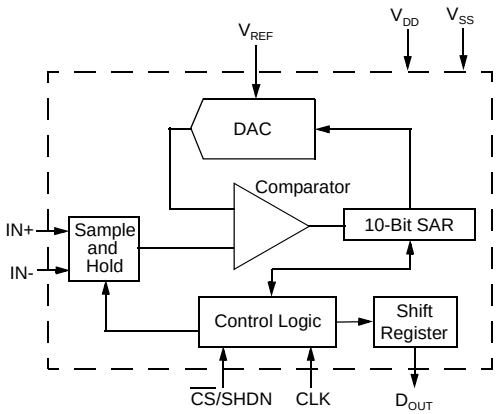


Illustration not to scale

Functional Block Diagram



SPI™ is a trademark of Motorola Inc.

1.0 ELECTRICAL CHARACTERISTICS

1.1 Maximum Ratings*

V_{DD}.....7.0V

All inputs and outputs w.r.t. V_{SS} ..... -0.6V to V_{DD} +0.6V

Storage temperature .....-65°C to +150°C

Ambient temp. with power applied .....-65°C to +125°C

ESD protection on all pins (HBM).....> 4kV

*Notice: Stresses above those listed under "Maximum ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operational listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

PIN FUNCTION TABLE

Name	Function
V_{DD}	+2.7V to 5.5V Power Supply
V_{SS}	Ground
IN+	Positive Analog Input
IN-	Negative Analog Input
CLK	Serial Clock
D_{OUT}	Serial Data Out
$\overline{CS}$/SHDN	$\overline{Chip\ Select}$/Shutdown Input
V_{REF}	Reference Voltage Input

ELECTRICAL CHARACTERISTICS

All parameters apply at V_{DD} = 5V, V_{SS} = 0V, V_{REF} = 5V, T_{AMB} = -40°C to +85°C, f_{SAMPLE} = 200 ksps and f_{CLK} = 14*f_{SAMPLE}, unless otherwise noted. Typical values apply for V_{DD} = 5V, T_{AMB} =25°C, unless otherwise noted.

Parameter	Sym	Min	Typ	Max	Units	Conditions
Conversion Rate:						
Conversion Time	t_{CONV}	—	—	10	clock cycles	
Analog Input Sample Time	t_{SAMPLE}		1.5		clock cycles	
Throughput Rate	f_{SAMPLE}	—	—	200 75	ksps ksps	V_{DD} = V_{REF} = 5V V_{DD} = V_{REF} = 2.7V
DC Accuracy:						
Resolution			10		bits	
Integral Nonlinearity	INL	—	±0.5	±1	LSB	
Differential Nonlinearity	DNL	—	±0.25	±1	LSB	No missing codes over temperature
Offset Error		—	—	±1.5	LSB	
Gain Error		—	—	±1	LSB	
Dynamic Performance:						
Total Harmonic Distortion	THD	—	-76	—	dB	V_{IN} = 0.1V to 4.9V@1 kHz
Signal to Noise and Distortion (SINAD)	SINAD	—	61	—	dB	V_{IN} = 0.1V to 4.9V@1 kHz
Spurious Free Dynamic Range	SFDR	—	80	—	dB	V_{IN} = 0.1V to 4.9V@1 kHz
Reference Input:						
Voltage Range	V_{REF}	0.25	—	V_{DD}	V	**Note 2**
Current Drain	I_{REF}	—	90 0.001	150 3	µA µA	 $\overline{CS}$ = V_{DD} = 5V

Note 1: This parameter is guaranteed by characterization and not 100% tested.

2: See graph that relates linearity performance to V_{REF} level.

3: Because the sample cap will eventually lose charge, clock rates below 10 kHz can affect linearity performance, especially at elevated temperatures.

All parameters apply at V_{DD} = 5V, V_{SS} = 0V, V_{REF} = 5V, T_{AMB} = -40°C to +85°C, f_{SAMPLE} = 200 ksps and f_{CLK} = 14*f_{SAMPLE}, unless otherwise noted. Typical values apply for V_{DD} = 5V, T_{AMB} =25°C, unless otherwise noted.

Parameter	Sym	Min	Typ	Max	Units	Conditions
Temperature Ranges:						
Specified Temperature Range	T_A	-40	—	+85	°C	
Operating Temperature Range	T_A	-40	—	+85	°C	
Storage Temperature Range	T_A	-65	—	+150	°C	
Thermal Package Resistance:						
Thermal Resistance, 8L-PDIP	θ_{JA}	—	85	—	°C/W	
Thermal Resistance, 8L-SOIC	θ_{JA}	—	163	—	°C/W	
Thermal Resistance, 8L-MSOP	θ_{JA}	—	206	—	°C/W	
Thermal Resistance, 8L-TSSOP	θ_{JA}	—		—	°C/W	
Analog Inputs:						
Input Voltage Range (IN+)	IN+	IN-	—	V_{REF}+IN-	V	
Input Voltage Range (IN-)	IN-	V_{SS}-100	—	V_{SS}+100	mV	
Leakage Current		—	0.001	±1	µA	
Switch Resistance	R_{SS}	—	1K	—	Ω	See Figure 4-1
Sample Capacitor	C_{SAMPLE}	—	20	—	pF	See Figure 4-1
Digital Input/Output:						
Data Coding Format		Straight Binary				
High Level Input Voltage	V_{IH}	0.7 V_{DD}	—	—	V	
Low Level Input Voltage	V_{IL}	—	—	0.3 V_{DD}	V	
High Level Output Voltage	V_{OH}	4.1	—	—	V	I_{OH} = -1 mA, V_{DD} = 4.5V
Low Level Output Voltage	V_{OL}	—	—	0.4	V	I_{OL} = 1 mA, V_{DD} = 4.5V
Input Leakage Current	I_{LI}	-10	—	10	µA	V_{IN} = V_{SS} or V_{DD}
Output Leakage Current	I_{LO}	-10	—	10	µA	V_{OUT} = V_{SS} or V_{DD}
Pin Capacitance (all inputs/outputs)	C_{IN}, C_{OUT}	—	—	10	pF	V_{DD} = 5.0V (**Note 1**) T_{AMB} = 25°C, f = 1 MHz
Timing Parameters:						
Clock Frequency	f_{CLK}	—	—	2.8 1.05	MHz MHz	V_{DD} = 5V (**Note 3**) V_{DD} = 2.7V (**Note 3**)
Clock High Time	t_{HI}	160	—	—	ns	
Clock Low Time	t_{LO}	160	—	—	ns	
$\overline{CS}$ Fall To First Rising CLK Edge	t_{SUCS}	100	—	—	ns	
CLK Fall To Output Data Valid	t_{DO}	—	—	125 200	ns ns	V_{DD} = 5V, See Figure 1-2 V_{DD} = 2.7, See Figure 1-2
CLK Fall To Output Enable	t_{EN}	—	—	125 200	ns ns	V_{DD} = 5V, See Figure 1-2 V_{DD} = 2.7, See Figure 1-2
$\overline{CS}$ Rise To Output Disable	t_{DIS}	—	—	100	ns	See test circuits, Figure 1-2 (**Note 1**)
$\overline{CS}$ Disable Time	t_{CSH}	350	—	—	ns	
D_{OUT} Rise Time	t_R	—	—	100	ns	See test circuits, Figure 1-2 (**Note 1**)
D_{OUT} Fall Time	t_F	—	—	100	ns	See test circuits, Figure 1-2 (**Note 1**)

Note 1: This parameter is guaranteed by characterization and not 100% tested.

2: See graph that relates linearity performance to V_{REF} level.

3: Because the sample cap will eventually lose charge, clock rates below 10 kHz can affect linearity performance, especially at elevated temperatures.

All parameters apply at V_{DD} = 5V, V_{SS} = 0V, V_{REF} = 5V, T_{AMB} = -40°C to +85°C, f_{SAMPLE} = 200 ksps and f_{CLK} = 14*f_{SAMPLE}, unless otherwise noted. Typical values apply for V_{DD} = 5V, T_{AMB} =25°C, unless otherwise noted.

Parameter	Sym	Min	Typ	Max	Units	Conditions
Power Requirements:						
Operating Voltage	V_{DD}	2.7	—	5.5	V	
Operating Current	I_{DD}	—	400 210	500	µA µA	V_{DD} = 5.0V, D_{OUT} unloaded V_{DD} = 2.7V, D_{OUT} unloaded
Standby Current	I_{DDS}	—	0.005	2	µA	$\overline{CS}$ = V_{DD} = 5.0V

Note 1: This parameter is guaranteed by characterization and not 100% tested.
2: See graph that relates linearity performance to V_{REF} level.
3: Because the sample cap will eventually lose charge, clock rates below 10 kHz can affect linearity performance, especially at elevated temperatures.

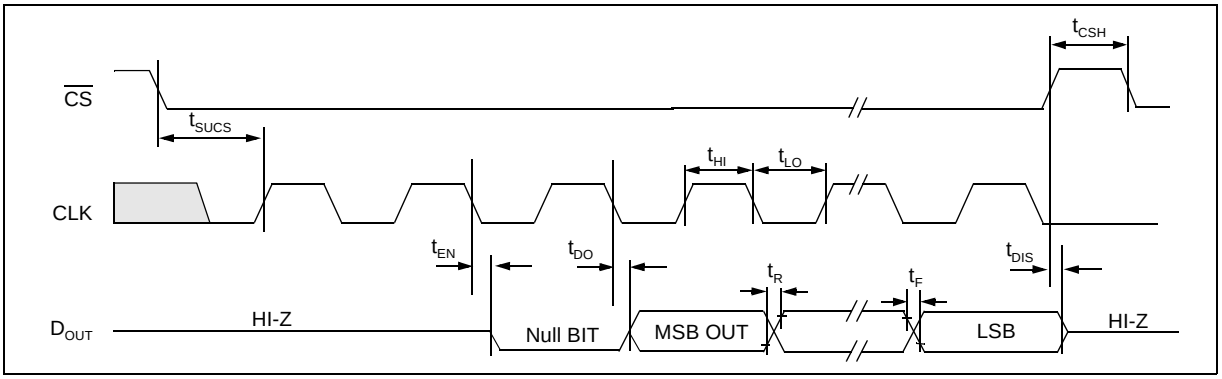
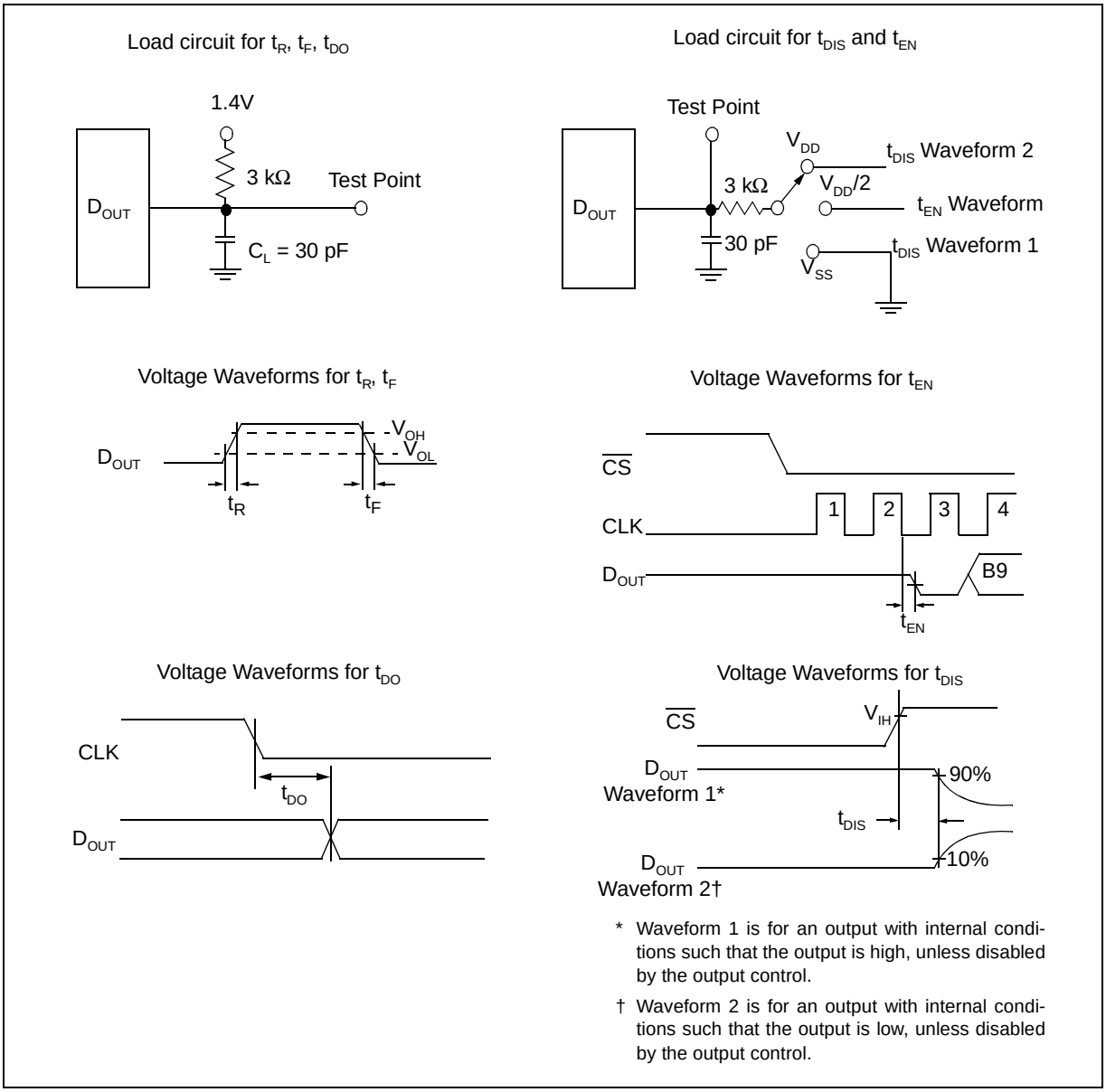


FIGURE 1-1: *Serial Timing.*



* Waveform 1 is for an output with internal conditions such that the output is high, unless disabled by the output control.

† Waveform 2 is for an output with internal conditions such that the output is low, unless disabled by the output control.

FIGURE 1-2: *Test Circuits.*

2.0 TYPICAL PERFORMANCE CHARACTERISTICS

Note: The graphs and tables provided following this note are a statistical summary based on a limited number of samples and are provided for informational purposes only. The performance characteristics listed herein are not tested or guaranteed. In some graphs or tables, the data presented may be outside the specified operating range (e.g., outside specified power supply range) and therefore outside the warranted range.

Note: Unless otherwise indicated, $V_{DD} = V_{REF} = 5V$, $f_{SAMPLE} = 200$ ksps, $f_{CLK} = 14$*Sample Rate, $T_A = 25°C$

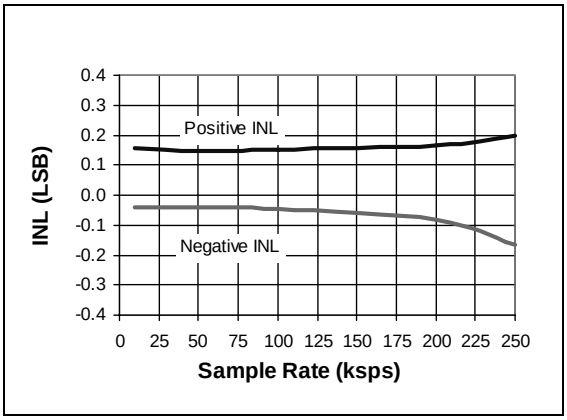


FIGURE 2-1: *Integral Nonlinearity (INL) vs. Sample Rate.*

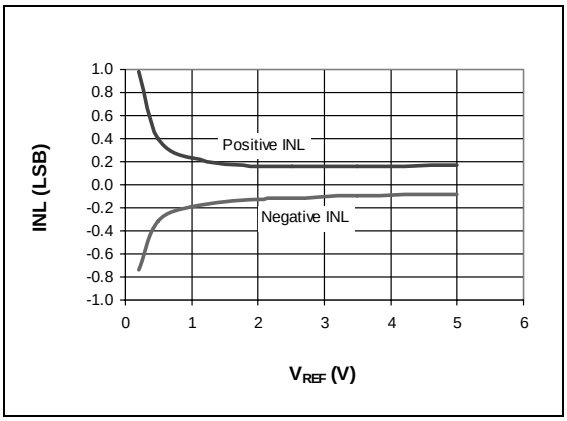


FIGURE 2-2: *Integral Nonlinearity (INL) vs. V_{REF}.*

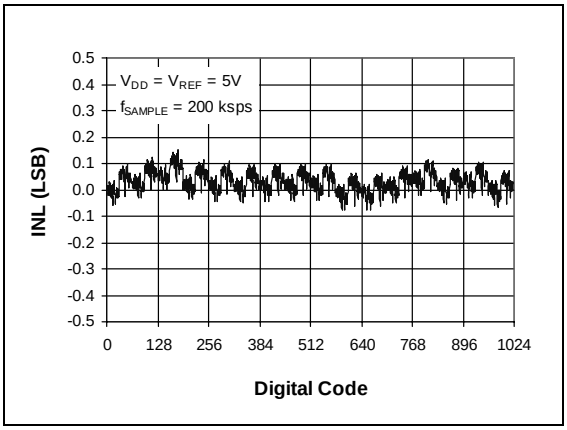


FIGURE 2-3: *Integral Nonlinearity (INL) vs. Code (Representative Part).*

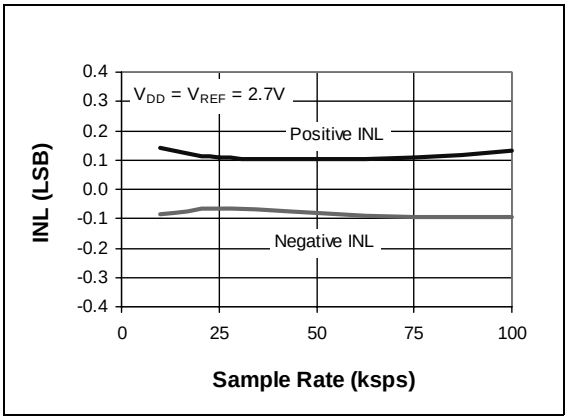


FIGURE 2-4: *Integral Nonlinearity (INL) vs. Sample Rate ($V_{DD} = 2.7V$).*

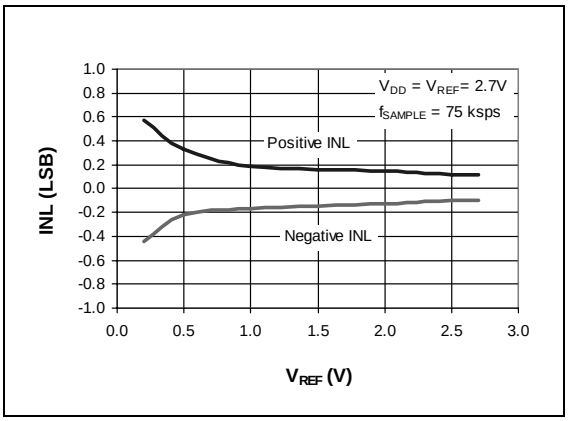


FIGURE 2-5: *Integral Nonlinearity (INL) vs. V_{REF} ($V_{DD} = 2.7V$).*

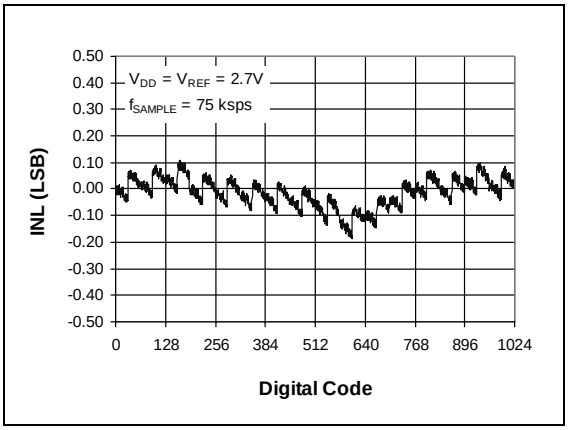


FIGURE 2-6: *Integral Nonlinearity (INL) vs. Code (Representative Part, $V_{DD} = 2.7V$).*

Note: Unless otherwise indicated, $V_{DD} = V_{REF} = 5V$, f_{SAMPLE} = 200 ksps, f_{CLK} = 14*Sample Rate,T_A = 25°C

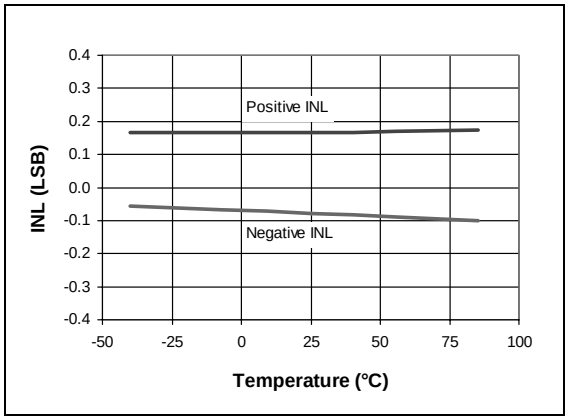


FIGURE 2-7: *Integral Nonlinearity (INL) vs. Temperature.*

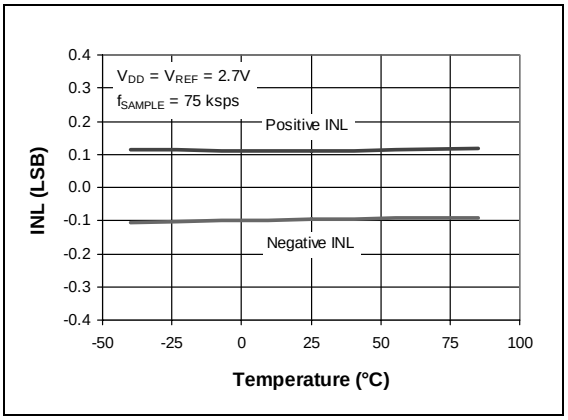


FIGURE 2-10: *Integral Nonlinearity (INL) vs. Temperature (V_{DD} = 2.7V).*

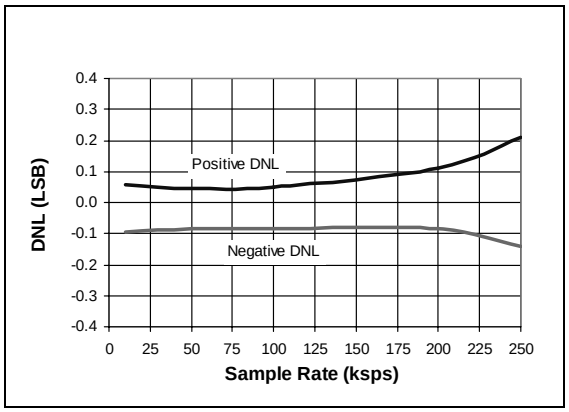


FIGURE 2-8: *Differential Nonlinearity (DNL) vs. Sample Rate.*

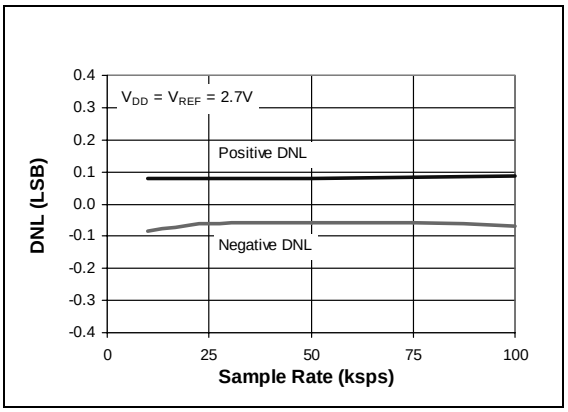


FIGURE 2-11: *Differential Nonlinearity (DNL) vs. Sample Rate (V_{DD} = 2.7V).*

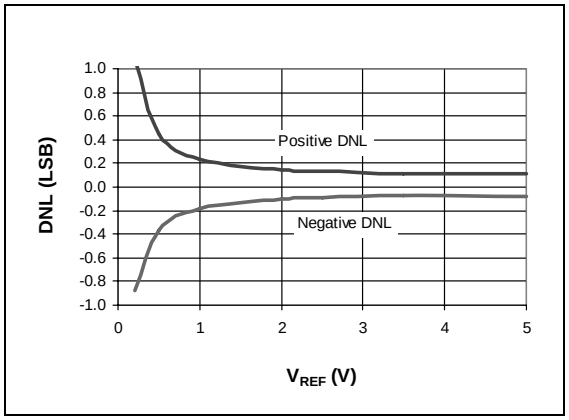


FIGURE 2-9: *Differential Nonlinearity (DNL) vs. V_{REF}.*

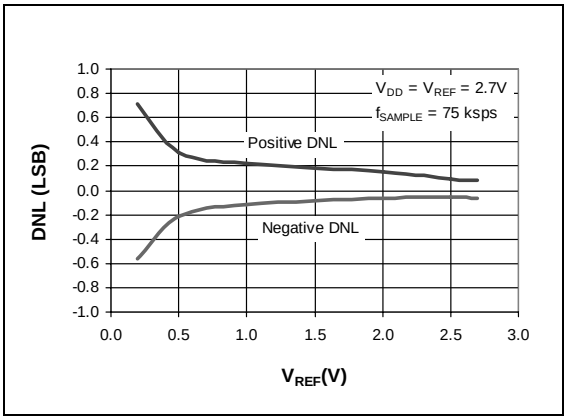


FIGURE 2-12: *Differential Nonlinearity (DNL) vs. V_{REF} (V_{DD} = 2.7V).*

Note: Unless otherwise indicated, $V_{DD} = V_{REF} = 5V$, $f_{SAMPLE} = 200$ ksps, $f_{CLK} = 14$*Sample Rate, $T_A = 25°C$

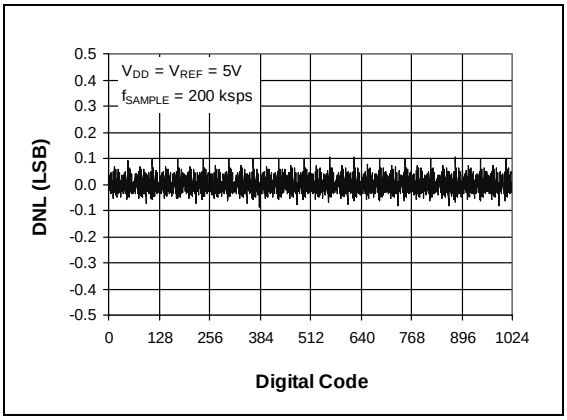


FIGURE 2-13: *Differential Nonlinearity (DNL) vs. Code (Representative Part).*

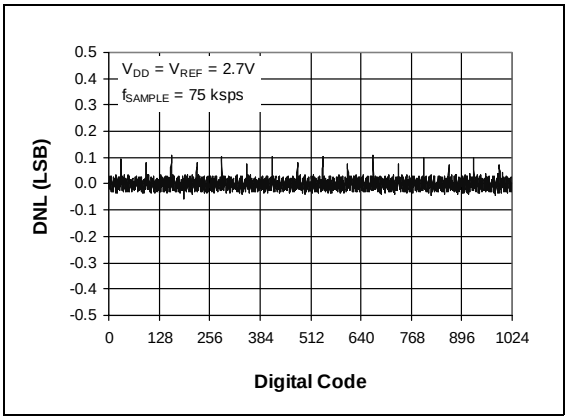


FIGURE 2-16: *Differential Nonlinearity (DNL) vs. Code (Representative Part, $V_{DD} = 2.7V$).*

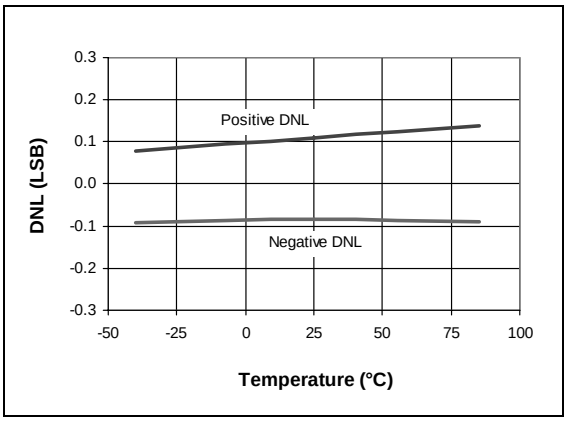


FIGURE 2-14: *Differential Nonlinearity (DNL) vs. Temperature.*

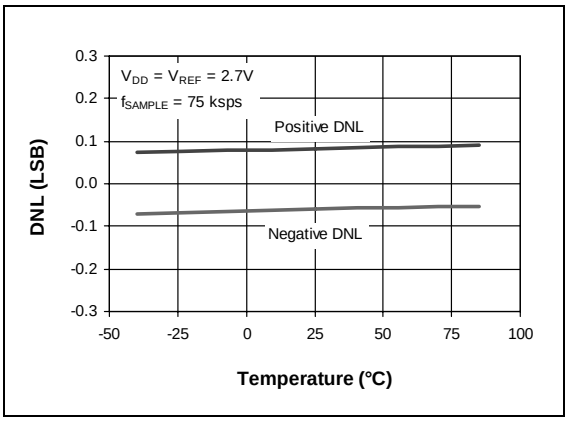


FIGURE 2-17: *Differential Nonlinearity (DNL) vs. Temperature ($V_{DD} = 2.7V$).*

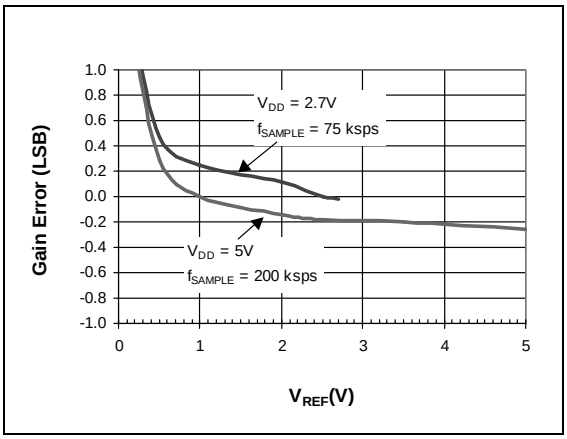


FIGURE 2-15: *Gain Error vs. V_{REF}.*

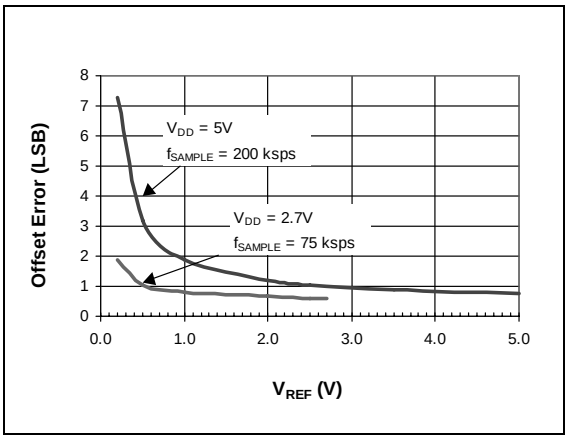


FIGURE 2-18: *Offset Error vs. V_{REF}.*

Note: Unless otherwise indicated, $V_{DD} = V_{REF} = 5V$, $f_{SAMPLE} = 200$ ksps, $f_{CLK} = 14$*Sample Rate, $T_A = 25°C$

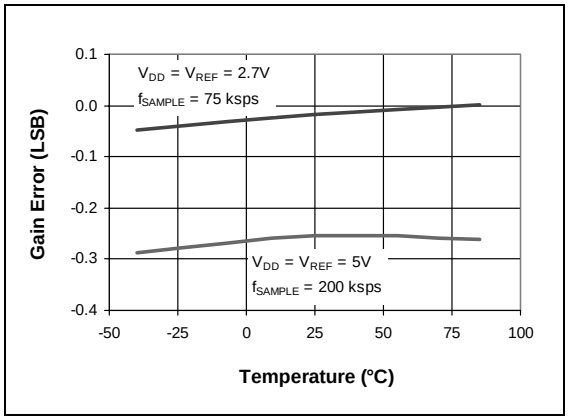


FIGURE 2-19: *Gain Error vs. Temperature.*

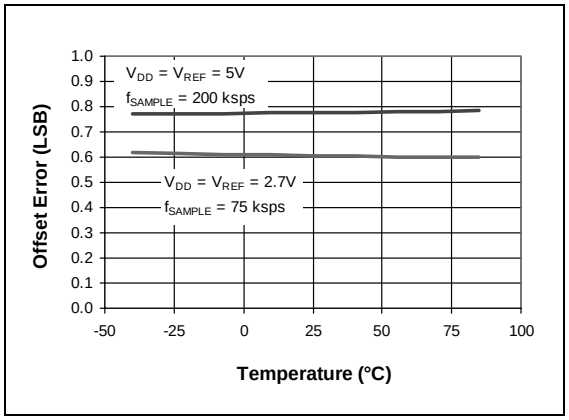


FIGURE 2-22: *Offset Error vs. Temperature.*

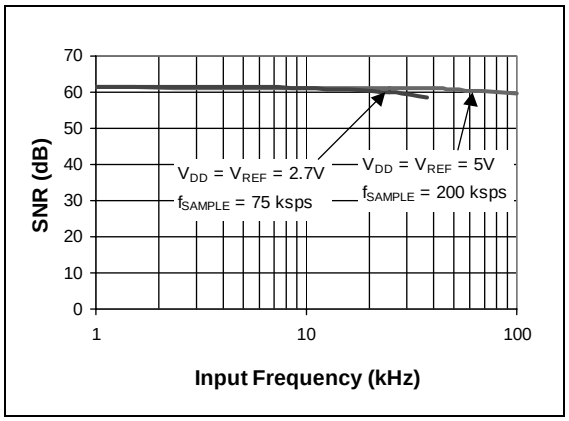


FIGURE 2-20: *Signal to Noise Ratio (SNR) vs. Input Frequency.*

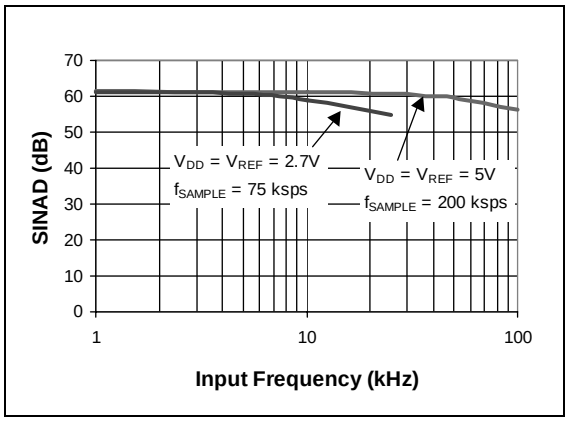


FIGURE 2-23: *Signal to Noise Ratio and Distortion (SINAD) vs. Input Frequency.*

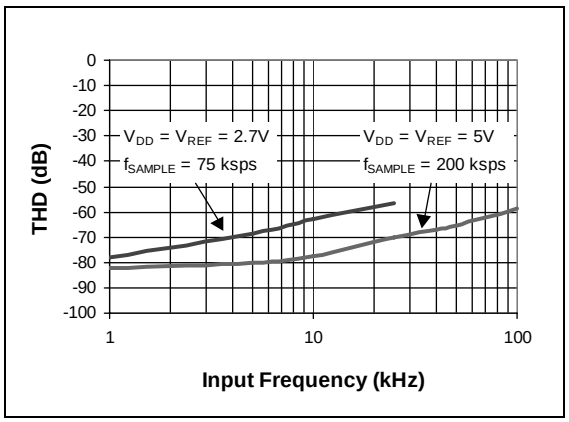


FIGURE 2-21: *Total Harmonic Distortion (THD) vs. Input Frequency.*

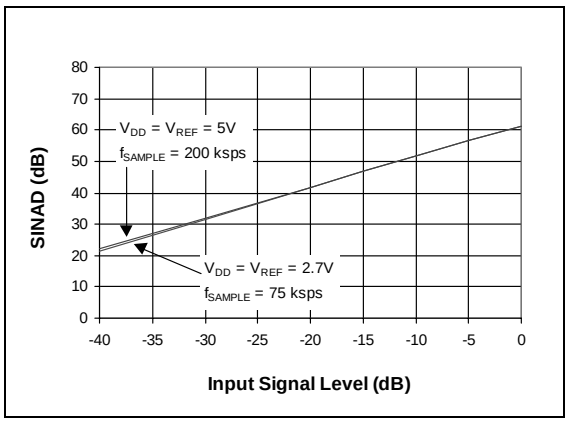


FIGURE 2-24: *Signal to Noise and Distortion (SINAD) vs. Input Signal Level.*

Note: Unless otherwise indicated, $V_{DD} = V_{REF} = 5V$, $f_{SAMPLE} = 200$ ksps, $f_{CLK} = 14$*Sample Rate, $T_A = 25°C$

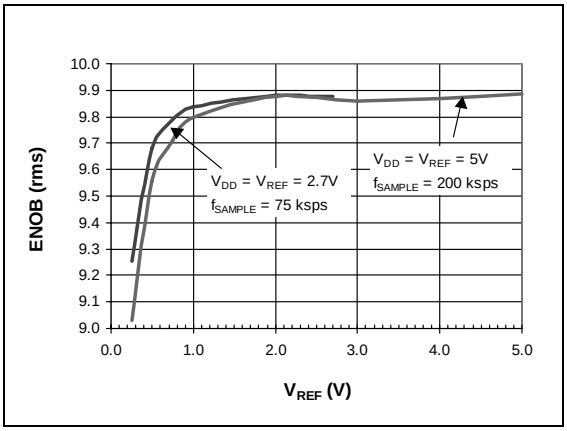


FIGURE 2-25: *Effective Number of Bits (ENOB) vs. V_{REF}.*

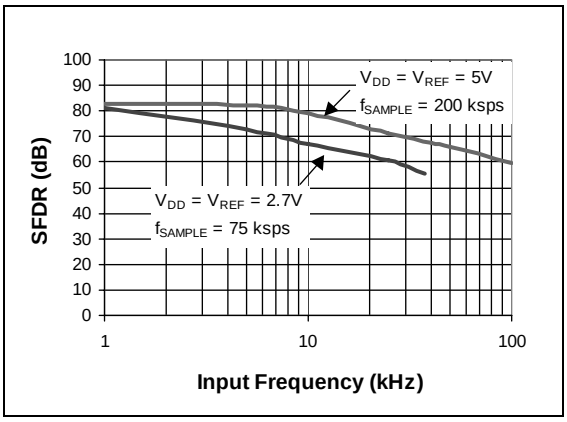


FIGURE 2-26: *Spurious Free Dynamic Range (SFDR) vs. Input Frequency.*

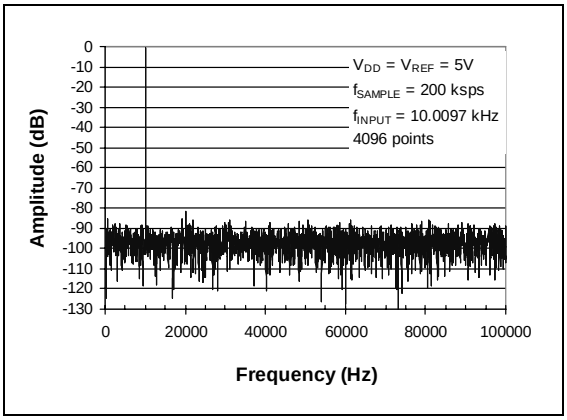


FIGURE 2-27: *Frequency Spectrum of 10 kHz Input (Representative Part).*

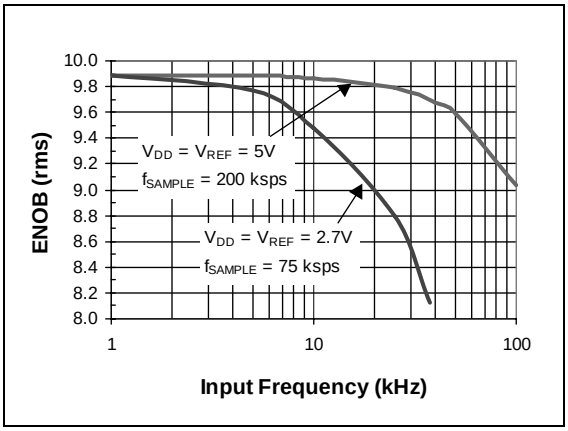


FIGURE 2-28: *Effective Number of Bits (ENOB) vs. Input Frequency.*

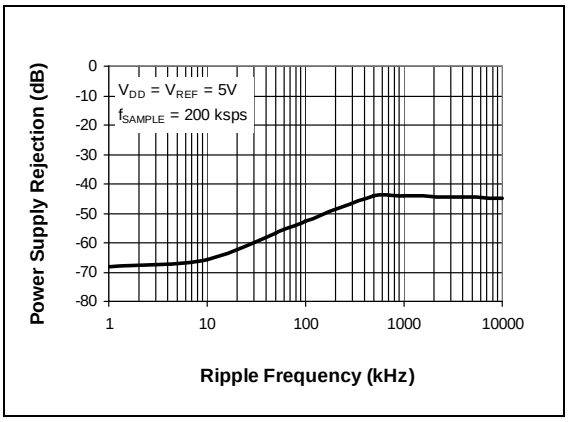


FIGURE 2-29: *Power Supply Rejection (PSR) vs. Ripple Frequency.*

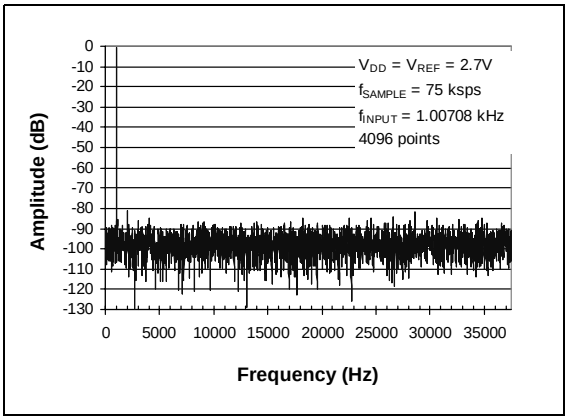


FIGURE 2-30: *Frequency Spectrum of 1 kHz Input (Representative Part, $V_{DD} = 2.7V$).*

Note: Unless otherwise indicated, $V_{DD} = V_{REF} = 5V$, $f_{SAMPLE} = 200$ ksps, $f_{CLK} = 14$*Sample Rate, $T_A = 25°C$

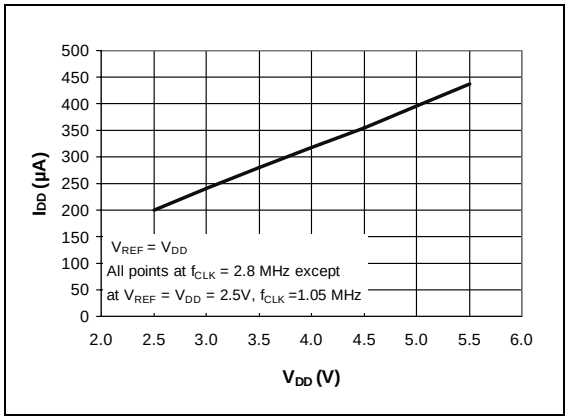


FIGURE 2-31: *I_{DD} vs. V_{DD}.*

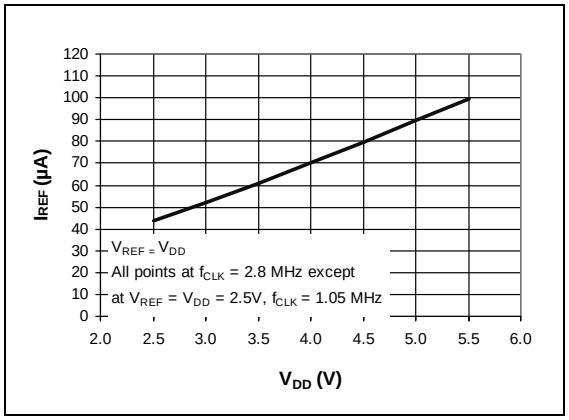


FIGURE 2-34: *I_{REF} vs. V_{DD}.*

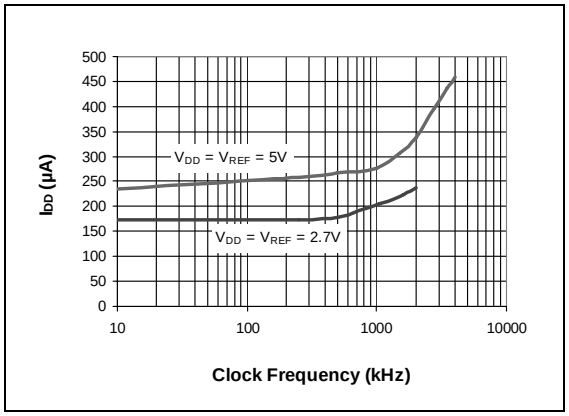


FIGURE 2-32: *I_{DD} vs. Clock Frequency.*

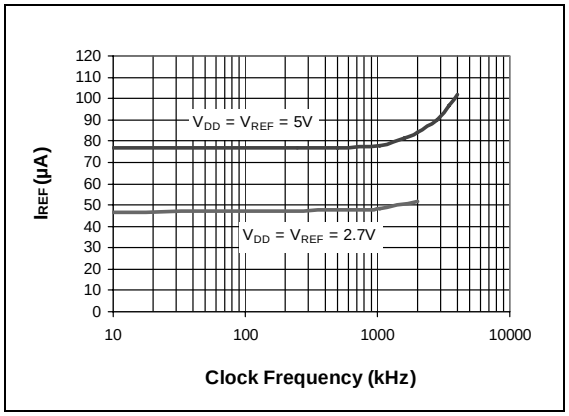


FIGURE 2-35: *I_{REF} vs. Clock Frequency.*

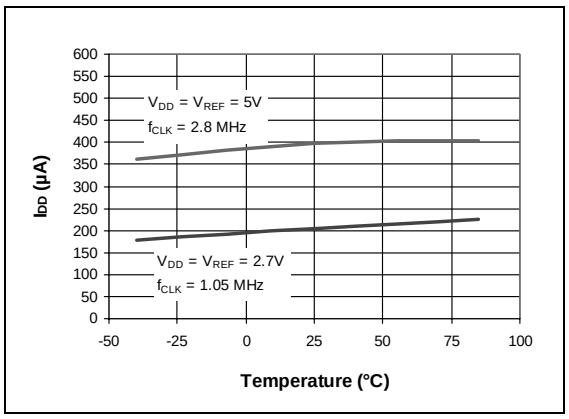


FIGURE 2-33: *I_{DD} vs. Temperature.*

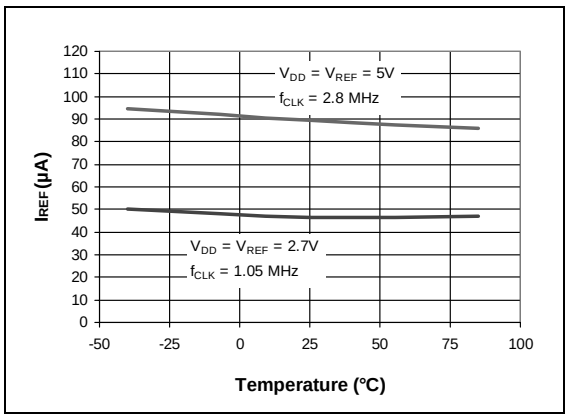


FIGURE 2-36: *I_{REF} vs. Temperature.*

Note: Unless otherwise indicated, $V_{DD} = V_{REF} = 5V$, $f_{SAMPLE} = 200$ ksps, $f_{CLK} = 14*$Sample Rate, $T_A = 25°C$

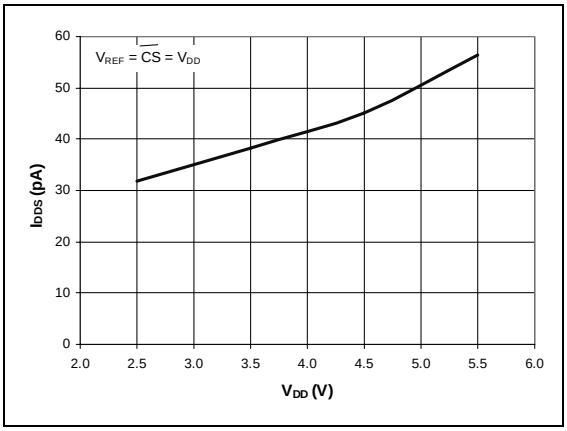


FIGURE 2-37: *I_{DDS} vs. V_{DD}.*

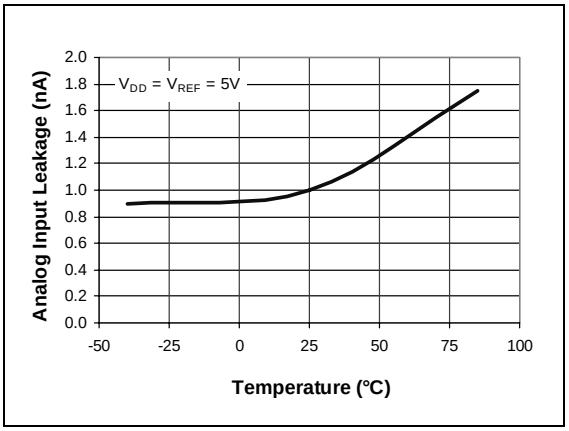


FIGURE 2-39: *Analog Input Leakage Current vs. Temperature.*

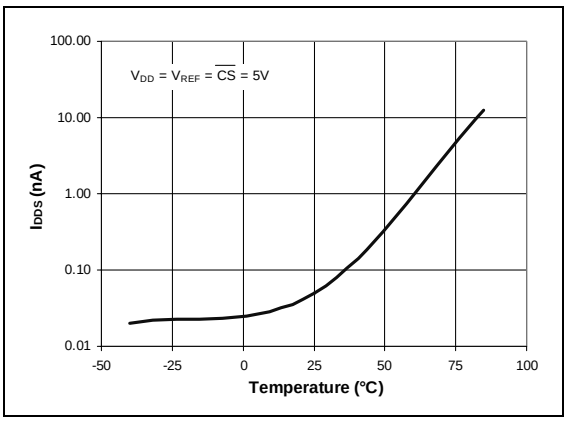


FIGURE 2-38: *I_{DDS} vs. Temperature.*

3.0 PIN DESCRIPTIONS

3.1 IN+

Positive analog input. This input can vary from IN- to V_{REF} + IN-.

3.2 IN-

Negative analog input. This input can vary ±100 mV from V_{SS}.

3.3 CS/SHDN(Chip Select/Shutdown)

The $\overline{CS}$/SHDN pin is used to initiate communication with the device when pulled low and will end a conversion and put the device in low power standby when pulled high. The $\overline{CS}$/SHDN pin must be pulled high between conversions.

3.4 CLK (Serial Clock)

The SPI clock pin is used to initiate a conversion and to clock out each bit of the conversion as it takes place. See Section 6.2 for constraints on clock speed.

3.5 DOUT (Serial Data output)

The SPI serial data output pin is used to shift out the results of the A/D conversion. Data will always change on the falling edge of each clock as the conversion takes place.

4.0 DEVICE OPERATION

The MCP3001 A/D converter employs a conventional SAR architecture. With this architecture, a sample is acquired on an internal sample/hold capacitor for 1.5 clock cycles starting on the first rising edge of the serial clock after $\overline{CS}$ has been pulled low. Following this sample time, the input switch of the converter opens and the device uses the collected charge on the internal sample and hold capacitor to produce a serial 10-bit digital output code. Conversion rates of 200 ksps are possible on the MCP3001. See Section 6.2 for information on minimum clock rates. Communication with the device is done using a 3-wire SPI-compatible interface.

4.1 Analog Inputs

The MCP3001 provides a single pseudo-differential input. The IN+ input can range from IN- to (V_{REF} +IN-). The IN- input is limited to ±100 mV from the V_{SS} rail. The IN- input can be used to cancel small signal common-mode noise which is present on both the IN+ and IN- inputs.

For the A/D Converter to meet specification, the charge holding capacitor, C_{SAMPLE} must be given enough time to acquire a 10-bit accurate voltage level during the 1.5 clock cycle sampling period. The analog input model is shown in Figure 4-1.

In this diagram, it is shown that the source impedance (R_S) adds to the internal sampling switch, (R_{SS}) impedance, directly affecting the time that is required to charge the capacitor, C_{SAMPLE}. Consequently, a larger source impedance increases the offset, gain, and integral linearity errors of the conversion.

Ideally, the impedance of the signal source should be near zero. This is achievable with an operational amplifier such as the MCP601, which has a closed loop output impedance of tens of ohms. The adverse affects of higher source impedances are shown in Figure 4-2.

If the voltage level of IN+ is equal to or less than IN-, the resultant code will be 000h. If the voltage at IN+ is equal to or greater than {[V_{REF} + (IN-)] - 1 LSB}, then the output code will be 3FFh. If the voltage level at IN- is more than 1 LSB below V_{SS}, then the voltage level at the IN+ input will have to go below V_{SS} to see the 000h output code. Conversely, if IN- is more than 1 LSB above Vss, then the 3FFh code will not be seen unless the IN+ input level goes above V_{REF} level.

4.2 Reference Input

The reference input (V_{REF}) determines the analog input voltage range and the LSB size, as shown below.

$$LSB\ Size = \frac{V_{REF}}{1024}$$

As the reference input is reduced, the LSB size is reduced accordingly. The theoretical digital output code produced by the A/D Converter is a function of the analog input signal and the reference input as shown below.

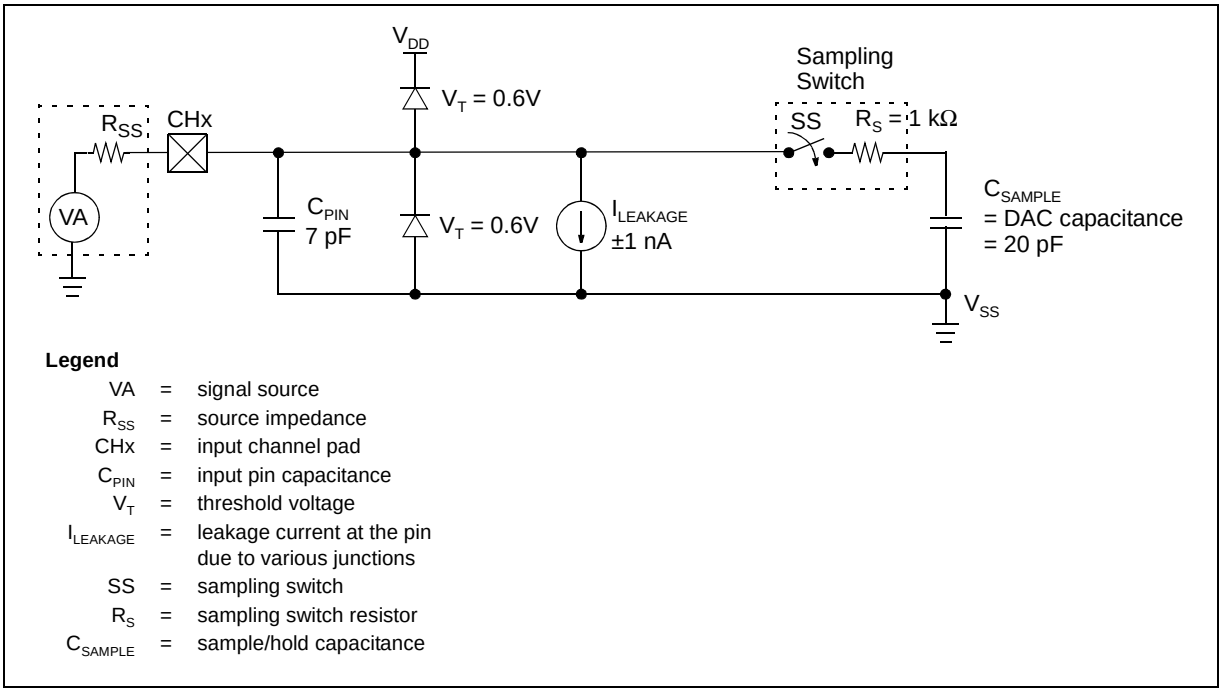
$$Digital\ Output\ Code = \frac{1024 * V_{IN}}{V_{REF}}$$

where:

V_{IN} = analog input voltage = V(IN+) - V(IN-)

V_{REF} = reference voltage

When using an external voltage reference device, the system designer should always refer to the manufacturer's recommendations for circuit layout. Any instability in the operation of the reference device will have a direct effect on the operation of the ADC.



Legend

VA	=	signal source
R_{SS}	=	source impedance
CHx	=	input channel pad
C_{PIN}	=	input pin capacitance
V_T	=	threshold voltage
$I_{LEAKAGE}$	=	leakage current at the pin due to various junctions
SS	=	sampling switch
R_S	=	sampling switch resistor
C_{SAMPLE}	=	sample/hold capacitance

FIGURE 4-1: *Analog Input Model.*

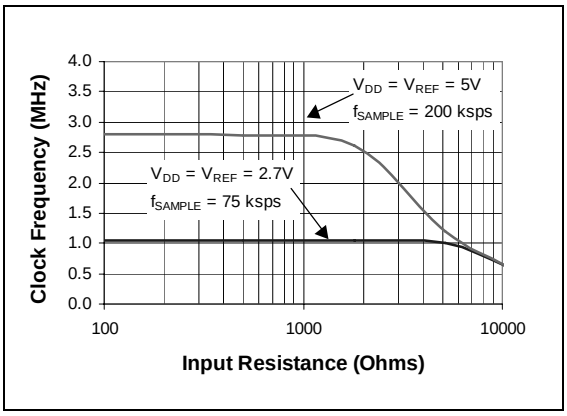
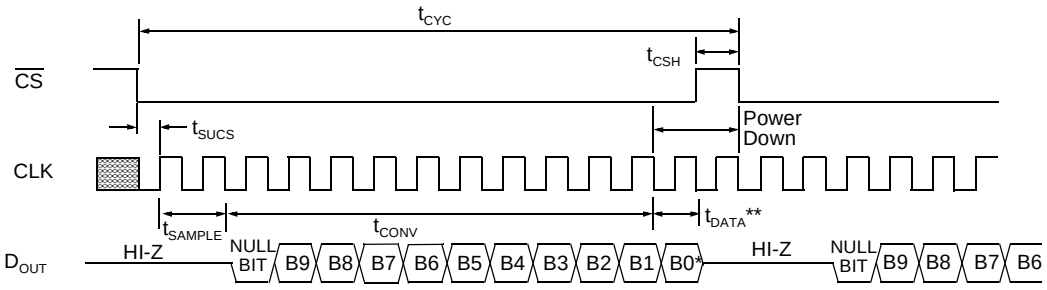


FIGURE 4-2: *Maximum Clock Frequency vs. Input Resistance (R_S) to maintain less than a 0.1LSB deviation in INL from nominal conditions.*

5.0 SERIAL COMMUNICATIONS

Communication with the device is done using a standard SPI compatible serial interface. Initiating communication with the MCP3001 begins with the $\overline{CS}$ going low. If the device was powered up with the $\overline{CS}$ pin low, it must be brought high and back low to initiate communication. The device will begin to sample the analog input on the first rising edge after $\overline{CS}$ goes low. The sample period will end in the falling edge of the second clock, at which time the device will output a low null bit. The next 10 clocks will output the result of the conversion with MSB first, as shown in Figure 5-1. Data is always output from the device on the falling edge of the clock. If all 10 data bits have been transmitted and the device continues to receive clocks while the $\overline{CS}$ is held low, the device will output the conversion result LSB first, as shown in Figure 5-2. If more clocks are provided to the device while $\overline{CS}$ is still low (after the LSB first data has been transmitted), the device will clock out zeros indefinitely.

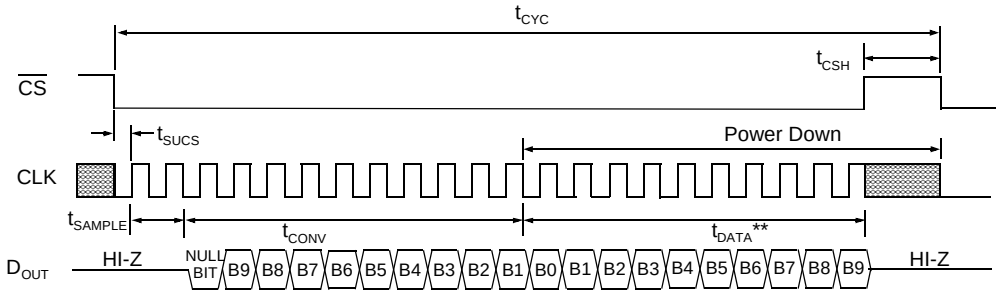
If it is desired, the $\overline{CS}$ can be raised to end the conversion period at any time during the transmission. Faster conversion rates can be obtained by using this technique if not all the bits are captured before starting a new cycle. Some system designers use this method by capturing only the highest order 8 bits and 'throwing away' the lower 2 bits.



* After completing the data transfer, if further clocks are applied with $\overline{CS}$ low, the ADC will output LSB first data, followed by zeros indefinitely. See Figure below.

** t_{DATA}: during this time, the bias current and the comparator powers down and the reference input becomes a high impedance node.

FIGURE 5-1: *Communication with MCP3001 (MSB first Format).*



* After completing the data transfer, if further clocks are applied with $\overline{CS}$ low, the ADC will output zeros indefinitely.

** t_{DATA}: during this time, the bias current and the comparator powers down and the reference input becomes a high impedance node leaving the CLK running to clock out the LSB-first data or zeros.

FIGURE 5-2: *Communication with MCP3001 (LSB first Format).*

6.0 APPLICATIONS INFORMATION

6.1 Using the MCP3001 with Microcontroller SPI Ports

With most microcontroller SPI ports, it is required to clock out eight bits at a time. If this is the case, it will be necessary to provide more clocks than are required for the MCP3001. As an example, Figure 6-1 and Figure 6-2 show how the MCP3001 can be interfaced to a microcontroller with a standard SPI port. Since the MCP3001 always clocks data out on the falling edge of clock, the MCU SPI port must be configured to match this operation. SPI Mode 0,0 (clock idles low) and SPI Mode 1,1 (clock idles high) are both compatible with the MCP3001. Figure 6-1 depicts the operation shown in SPI Mode 0,0, which requires that the CLK from the microcontroller idles in the 'low' state. As shown in the diagram, the MSB is clocked out of the ADC on the falling edge of the third clock pulse. After the first eight clocks have been sent to the device, the microcontroller's receive buffer will contain two unknown bits (the output is at high impedance for the first two clocks), the null bit and the highest order five bits of the conversion. After the second eight clocks have been sent to the device, the MCU receive register will contain the lowest order five bits and the B1-B4 bits repeated as the ADC has begun to shift out LSB first data with the extra clocks. Typical procedure would then call for the lower order byte of data to be shifted right by three bits to remove the extra B1-B4 bits. The B9-B5 bits are then rotated 3 bits to the right with B7-B5 rotating from the high order byte to the lower order byte. Easier manipulation of the converted data can be obtained by using this method.

Figure 6-2 shows SPI Mode 1,1 communication which requires that the clock idles in the high state. As with mode 0,0, the ADC outputs data on the falling edge of the clock and the MCU latches data from the ADC in on the rising edge of the clock.

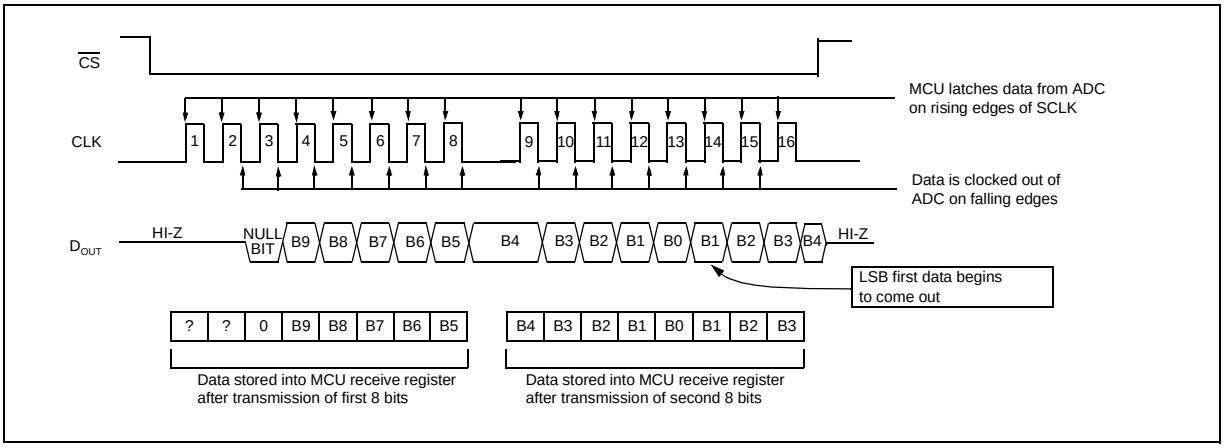


FIGURE 6-1: *SPI Communication with the MCP3001 using 8-bit segments (Mode 0,0: SCLK idles low).*

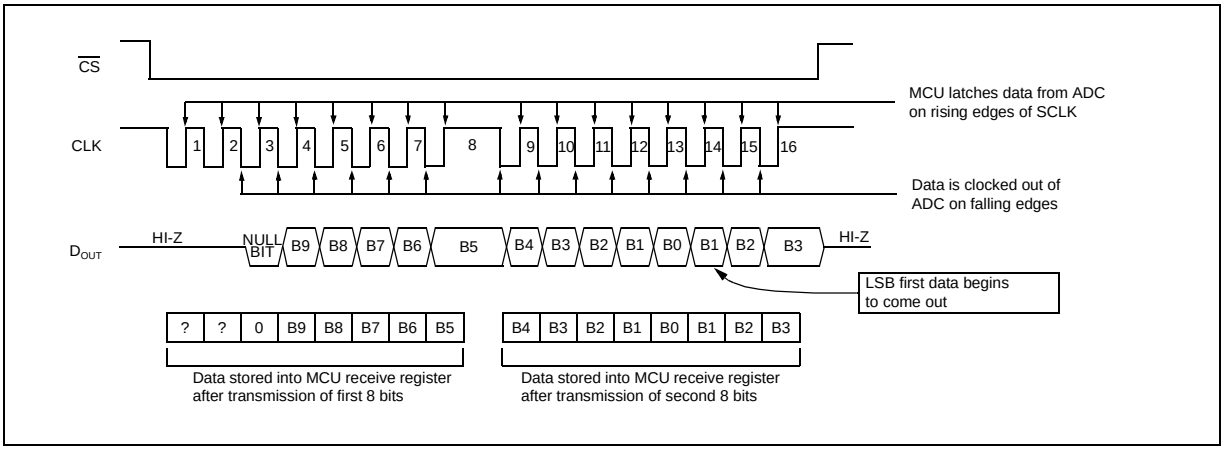


FIGURE 6-2: *SPI Communication with the MCP3001 using 8-bit segments (Mode 1,1: SCLK idles high).*

6.2 Maintaining Minimum Clock Speed

When the MCP3001 initiates the sample period, charge is stored on the sample capacitor. When the sample period is complete, the device converts one bit for each clock that is received. It is important for the user to note that a slow clock rate will allow charge to bleed off the sample cap while the conversion is taking place. At 85°C (worst case condition), the part will maintain proper charge on the sample cap for 700 µs at V_{DD} = 2.7V and 1.5 ms at V_{DD} = 5V. This means that at V_{DD} = 2.7V, the time it takes to transmit the first 14 clocks must not exceed 700 µs. Failure to meet this criterion may induce linearity errors into the conversion outside the rated specifications.

6.3 Buffering/Filtering the Analog Inputs

If the signal source for the ADC is not a low impedance source, it will have to be buffered or inaccurate conversion results may occur. See Figure 4-2. It is also recommended that a filter be used to eliminate any signals that may be aliased back into the conversion results. This is illustrated in Figure 6-3 where an op amp is used to drive, filter and gain the analog input of the MCP3001. This amplifier provides a low impedance source for the converter input and a low pass filter, which eliminates unwanted high frequency noise.

Low pass (anti-aliasing) filters can be designed using Microchip's interactive FilterLab™ software. FilterLab will calculate capacitor and resistor values, as well as determine the number of poles that are required for the application. For more information on filtering signals, see the application note AN699 *"Anti-Aliasing Analog Filters for Data Acquisition Systems."*

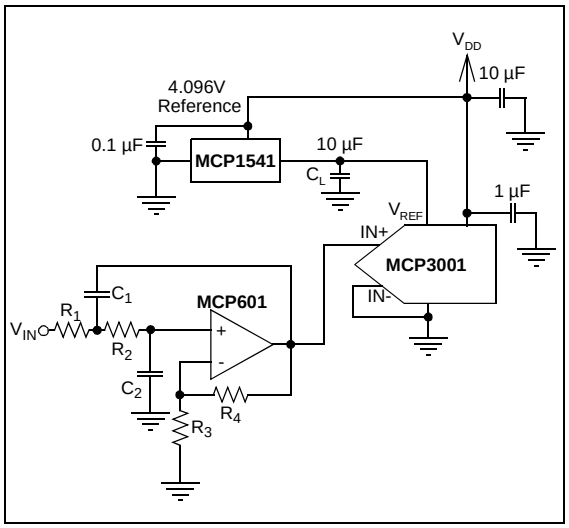


FIGURE 6-3: *The MCP601 operational amplifier is used to implement a 2nd order anti-aliasing filter for the signal being converted by the MCP3001.*

6.4 Layout Considerations

When laying out a printed circuit board for use with analog components, care should be taken to reduce noise wherever possible. A bypass capacitor should always be used with this device and should be placed as close as possible to the device pin. A bypass capacitor value of 1 µF is recommended.

Digital and analog traces should be separated as much as possible on the board and no traces should run underneath the device or the bypass capacitor. Extra precautions should be taken to keep traces with high frequency signals (such as clock lines) as far as possible from analog traces.

Use of an analog ground plane is recommended in order to keep the ground potential the same for all devices on the board. Providing V_{DD} connections to devices in a "star" configuration can also reduce noise by eliminating current return paths and associated errors. See Figure 6-4. For more information on layout tips when using ADC, refer to AN-688 *"Layout Tips for 12-Bit A/D Converter Applications"*.

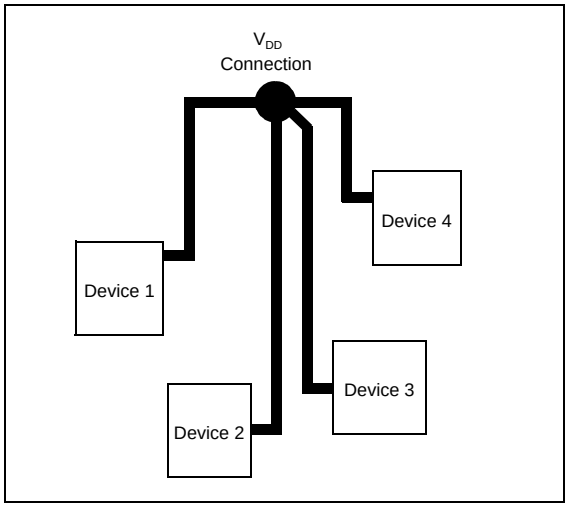
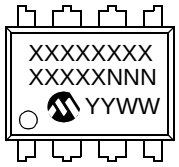


FIGURE 6-4: *V_{DD} traces arranged in a 'Star' configuration in order to reduce errors caused by current return paths.*

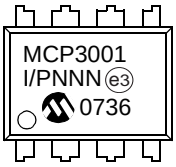
7.0 PACKAGING INFORMATION

7.1 Package Marking Information

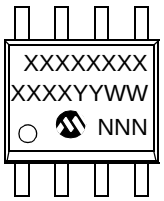
8-Lead PDIP (300 mil)



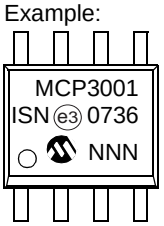
Example:



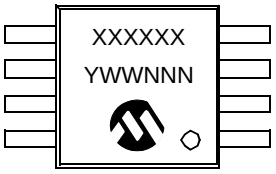
8-Lead SOIC (150 mil)



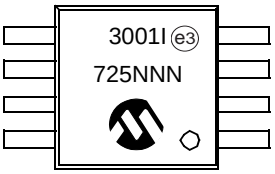
Example:



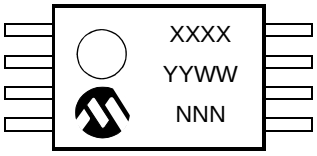
8-Lead MSOP



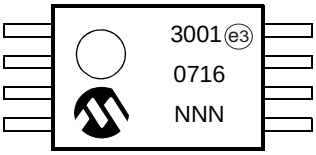
Example:



8-Lead TSSOP



Example:

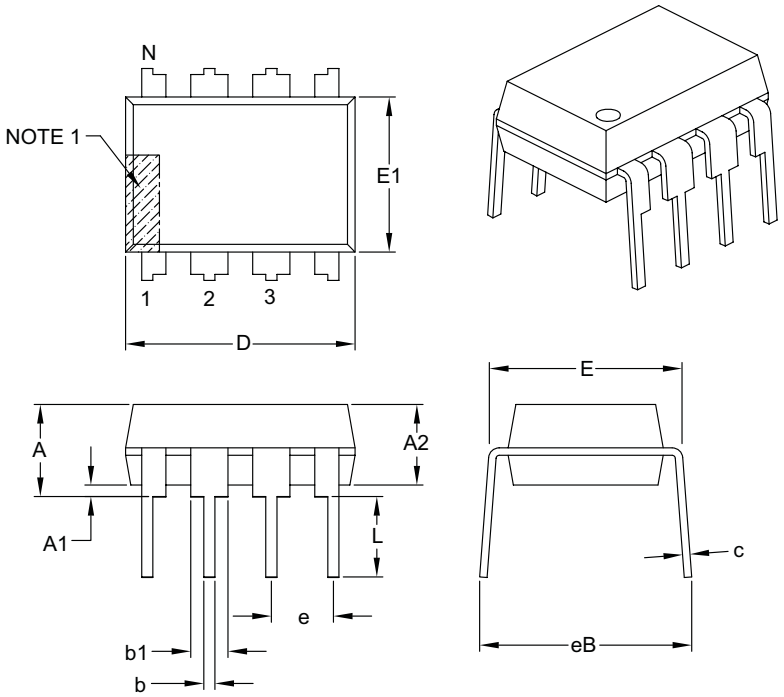


Legend:	XX...X	Customer-specific information
	Y	Year code (last digit of calendar year)
	YY	Year code (last 2 digits of calendar year)
	WW	Week code (week of January 1 is week ‘01’)
	NNN	Alphanumeric traceability code
	(e3)	Pb-free JEDEC designator for Matte Tin (Sn)
	*	This package is Pb-free. The Pb-free JEDEC designator ((e3)) can be found on the outer packaging for this package.

Note: In the event the full Microchip part number cannot be marked on one line, it will be carried over to the next line, thus limiting the number of available characters for customer-specific information.

8-Lead Plastic Dual In-Line (P) – 300 mil Body [PDIP]

Note: For the most current package drawings, please see the Microchip Packaging Specification located at http://www.microchip.com/packaging



Units		INCHES		
Dimension Limits		MIN	NOM	MAX
Number of Pins	N		8	
Pitch	e		.100 BSC	
Top to Seating Plane	A	–	–	.210
Molded Package Thickness	A2	.115	.130	.195
Base to Seating Plane	A1	.015	–	–
Shoulder to Shoulder Width	E	.290	.310	.325
Molded Package Width	E1	.240	.250	.280
Overall Length	D	.348	.365	.400
Tip to Seating Plane	L	.115	.130	.150
Lead Thickness	c	.008	.010	.015
Upper Lead Width	b1	.040	.060	.070
Lower Lead Width	b	.014	.018	.022
Overall Row Spacing §	eB	–	–	.430

Notes:

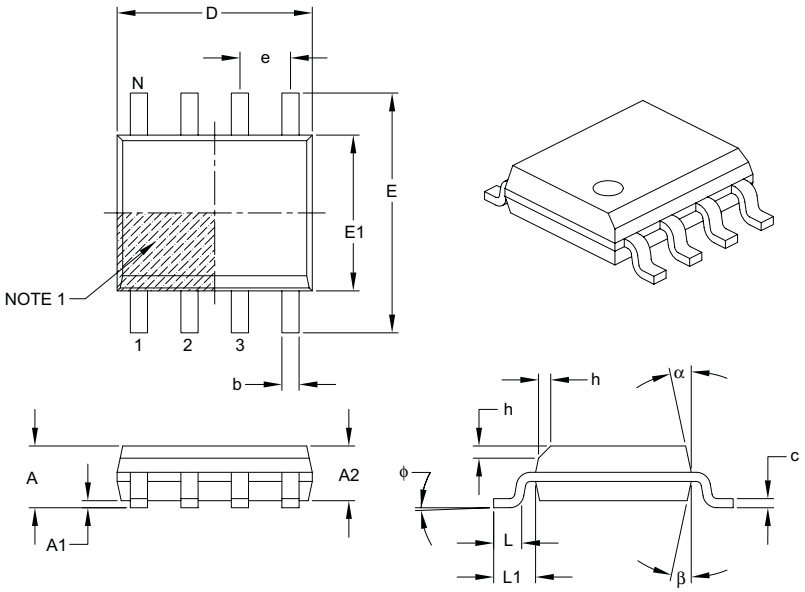
1. Pin 1 visual index feature may vary, but must be located with the hatched area.
2. § Significant Characteristic.
3. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010" per side.
4. Dimensioning and tolerancing per ASME Y14.5M.

   BSC: Basic Dimension. Theoretically exact value shown without tolerances.

Microchip Technology Drawing C04-018B

8-Lead Plastic Small Outline (SN) – Narrow, 3.90 mm Body [SOIC]

Note: For the most current package drawings, please see the Microchip Packaging Specification located at http://www.microchip.com/packaging



Units		MILLMETERS		
Dimension Limits		MIN	NOM	MAX
Number of Pins	N		8	
Pitch	e		1.27 BSC	
Overall Height	A	–	–	1.75
Molded Package Thickness	A2	1.25	–	–
Standoff §	A1	0.10	–	0.25
Overall Width	E		6.00 BSC	
Molded Package Width	E1		3.90 BSC	
Overall Length	D		4.90 BSC	
Chamfer (optional)	h	0.25	–	0.50
Foot Length	L	0.40	–	1.27
Footprint	L1		1.04 REF	
Foot Angle	φ	0°	–	8°
Lead Thickness	c	0.17	–	0.25
Lead Width	b	0.31	–	0.51
Mold Draft Angle Top	α	5°	–	15°
Mold Draft Angle Bottom	β	5°	–	15°

Notes:

1. Pin 1 visual index feature may vary, but must be located within the hatched area.
2. § Significant Characteristic.
3. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.15 mm per side.
4. Dimensioning and tolerancing per ASME Y14.5M.

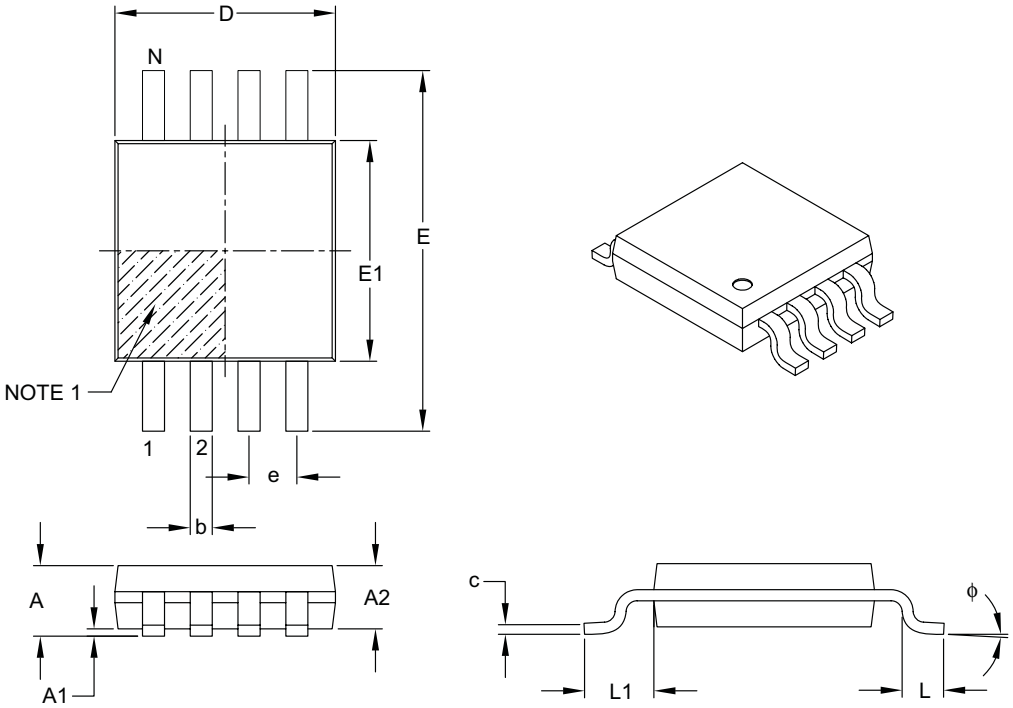
   BSC: Basic Dimension. Theoretically exact value shown without tolerances.

   REF: Reference Dimension, usually without tolerance, for information purposes only.

Microchip Technology Drawing C04-057B

8-Lead Plastic Micro Small Outline Package (MS) [MSOP]

Note: For the most current package drawings, please see the Microchip Packaging Specification located at http://www.microchip.com/packaging



Units		MILLIMETERS		
Dimension Limits		MIN	NOM	MAX
Number of Pins	N		8	
Pitch	e		0.65 BSC	
Overall Height	A	–	–	1.10
Molded Package Thickness	A2	0.75	0.85	0.95
Standoff	A1	0.00	–	0.15
Overall Width	E		4.90 BSC	
Molded Package Width	E1		3.00 BSC	
Overall Length	D		3.00 BSC	
Foot Length	L	0.40	0.60	0.80
Footprint	L1		0.95 REF	
Foot Angle	ϕ	0°	–	8°
Lead Thickness	c	0.08	–	0.23
Lead Width	b	0.22	–	0.40

Notes:

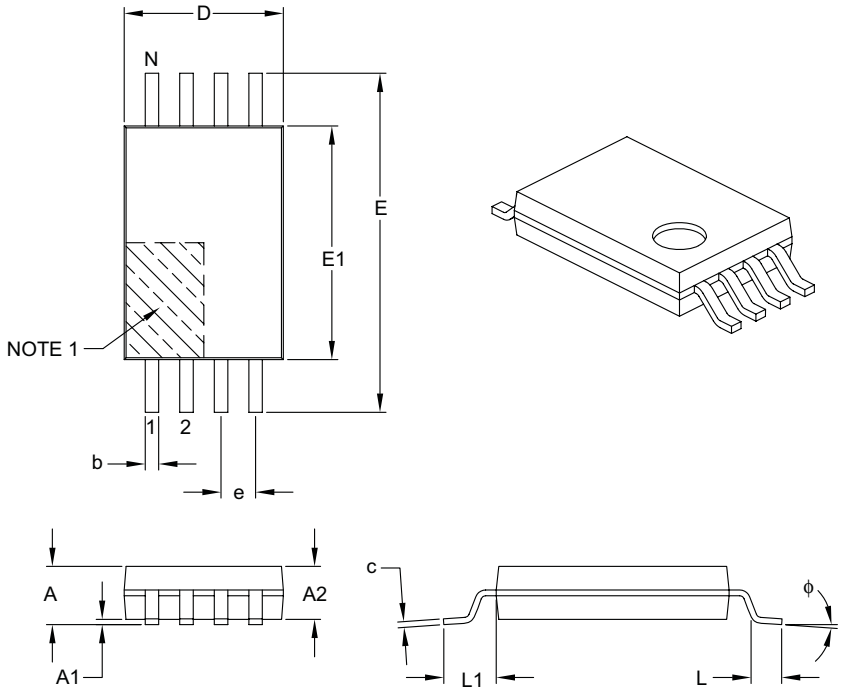
1. Pin 1 visual index feature may vary, but must be located within the hatched area.
2. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.15 mm per side.
3. Dimensioning and tolerancing per ASME Y14.5M.

BSC: Basic Dimension. Theoretically exact value shown without tolerances.
REF: Reference Dimension, usually without tolerance, for information purposes only.

Microchip Technology Drawing C04-111B

8-Lead Plastic Thin Shrink Small Outline (ST) – 4.4 mm Body [TSSOP]

Note: For the most current package drawings, please see the Microchip Packaging Specification located at http://www.microchip.com/packaging



Units		MILLIMETERS		
Dimension Limits		MIN	NOM	MAX
Number of Pins	N		8	
Pitch	e		0.65 BSC	
Overall Height	A	–	–	1.20
Molded Package Thickness	A2	0.80	1.00	1.05
Standoff	A1	0.05	–	0.15
Overall Width	E		6.40 BSC	
Molded Package Width	E1	4.30	4.40	4.50
Molded Package Length	D	2.90	3.00	3.10
Foot Length	L	0.45	0.60	0.75
Footprint	L1		1.00 REF	
Foot Angle	ϕ	0°	–	8°
Lead Thickness	c	0.09	–	0.20
Lead Width	b	0.19	–	0.30

Notes:

1. Pin 1 visual index feature may vary, but must be located within the hatched area.
2. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.15 mm per side.
3. Dimensioning and tolerancing per ASME Y14.5M.

BSC: Basic Dimension. Theoretically exact value shown without tolerances.
REF: Reference Dimension, usually without tolerance, for information purposes only.

Microchip Technology Drawing C04-086B

CUI DEVICES

SERIES: CFM-40 | DESCRIPTION: DC AXIAL FAN

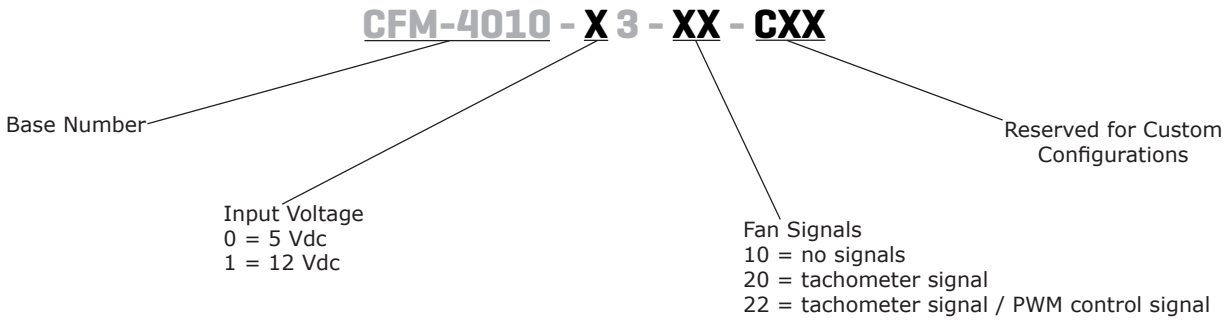
FEATURES

- 40 x 40 mm frame
- high fan speed for greater air flow
- dual ball bearing construction
- auto restart protection standard on all models

MODEL	input voltage		input current		input power	rated speed	air flow[1]	static pressure[2]	noise
	rated (Vdc)	range (Vdc)	typ (A)	max (A)	max (W)	typ (RPM)	(CFM)	(inch H_2O)	max (dBA)
CFM-4010-03	5	4~5.75	0.16	0.24	1.2	8,300	10.0	0.29	37.0
CFM-4010-13	12	6~13.8	0.10	0.13	1.56	8,300	10.0	0.29	37.0

Notes: 1. At 0 inch H_20 static pressure.
2. At 0 CFM airflow.

PART NUMBER KEY



INPUT

parameter	conditions/description	min	typ	max	units
operating input voltage	5 Vdc input models 12 Vdc input models	4 6	5 12	5.75 13.8	Vdc Vdc
current	5 Vdc input models 12 Vdc input models		0.16 0.10	0.24 0.13	A A
power	5 Vdc input models 12 Vdc input models		0.80 1.20	1.2 1.56	W W
starting voltage	at 25°C 5 Vdc input models 12 Vdc input models		 4 6		 Vdc Vdc

PERFORMANCE

parameter	conditions/description	min	typ	max	units
rated speed	at 25°C, after 10 minutes	7,470	8,300	9,130	RPM
air flow	at 0 inch H_2O, see performance curves		10.00		CFM
static pressure	at 0 CFM, see performance curves		0.29		inch H_2O
noise	at 1 m		35.5	37.0	dBA

PROTECTIONS / SIGNALS[1]

parameter	conditions/description	min	typ	max	units
auto restart protection	available on all models				
tachometer signal	available on "20" and "22" models				
PWM control signal	available on "22" models				

Notes: 1. See application notes for details.

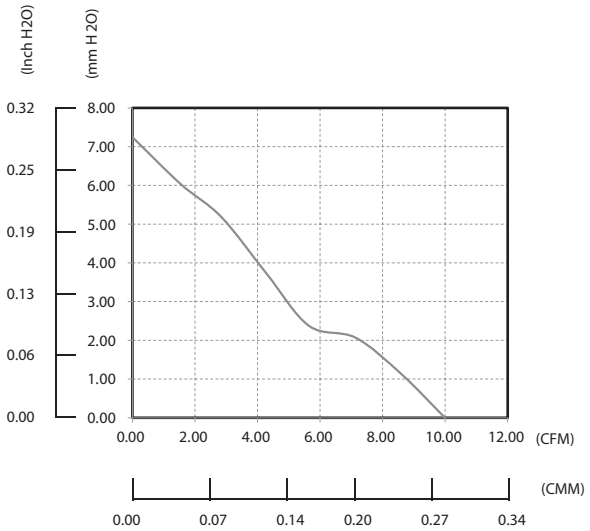
SAFETY & COMPLIANCE

parameter	conditions/description	min	typ	max	units
insulation resistance of frame	at 500 Vdc between frame and positive terminal	10			MΩ
dielectric strength	at 500 Vac, 60 Hz, 1 minute between frame and positive terminal			5	mA
safety approvals	UL/cUL 507, TUV (EN 62368-1)				
EMI/EMC	EN 55022:2010+AC:2011 Class B, EN 61000-3-2:2014, EN 61000-3-3:2013, EN 55024:2010				
life expectancy	at 45°C, 15~65% RH		70,000		hours
RoHS	yes				

ENVIRONMENTAL

parameter	conditions/description	min	typ	max	units
operating temperature		-10		70	°C
storage temperature		-40		70	°C
operating humidity	non-condensing	5		90	%
storage humidity	non-condensing	5		95	%

PERFORMANCE CURVES



MECHANICAL

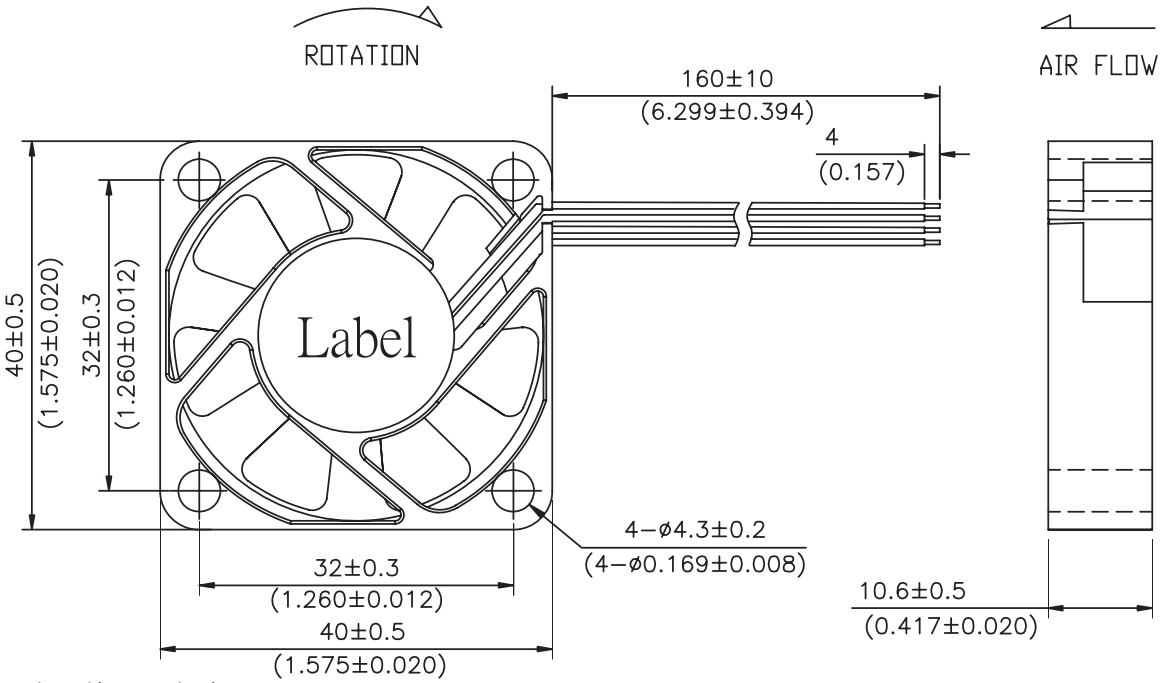
parameter	conditions/description	min	typ	max	units
motor	4 pole DC brushless				
bearing system	ball bearing				
direction of rotation	counter-clockwise viewed from front of fan blade				
dimensions	40 x 40 x 10.6				mm
material	PBT (UL94V-0)				
weight	5 Vdc input models 12 Vdc input models		13.5 14.2		g g

MECHANICAL DRAWING

units: mm [inch]

wire: UL 1061, 28 AWG

WIRE CONNECTIONS	
Wire Color	Function
Red	+Vin
Black	-Vin
Yellow[1]	FG Signal
Blue[1]	PWM



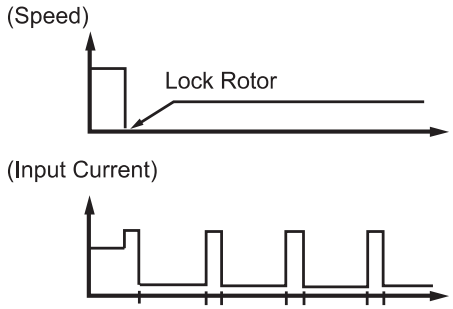
Note: 1. Wires only present on versions with output signals.

APPLICATION NOTES

Auto Restart Protection/Current Limit Protection

When the fan motor is locked, the device will cut off the drive current within two to six seconds and restart automatically after a few seconds. If the lock situation is continued, the device will work on a repeated cycle of cut-off and restart until the lock is released. (See Figure 1 below).

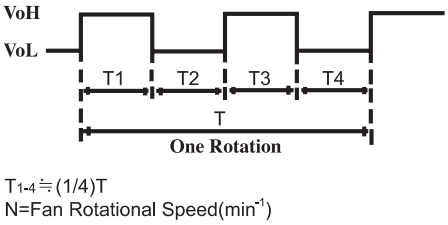
Figure 1 Current Limit Protection



Pulse Sensor/Tachometer Signal/FG

Pulse Sensor is for detecting the rotational speed of the fan motor. At locked rotor condition, the signal stops cycling and the output is fixed at VoH or VoL (See Figures 2~3 below).

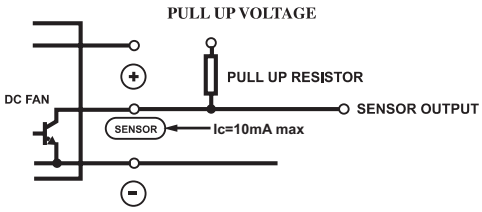
Figure 2 Output Waveform



$T_{1-4} \fallingdotseq (1/4)T$

N=Fan Rotational Speed(min^{-1})

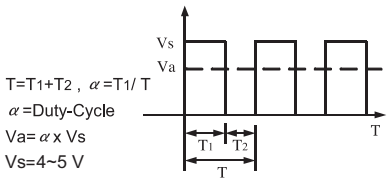
Figure 3 FG Signal Output Circuit: Open Collector



PMW Control Signal

A speed control lead can be provided that will accept a PWM signal from the customer circuit to vary the speed of the fan. The change in speed is linear by changing the Duty-Cycle of the PWM. Open collector type and pull-up voltage is changed by maximum operating voltage and sink current by consuming current. (See Figure 4 below).

Figure 4 Duty Cycle



$T=T_1+T_2$, $\alpha = T_1/T$

α =Duty-Cycle

Va= α x Vs

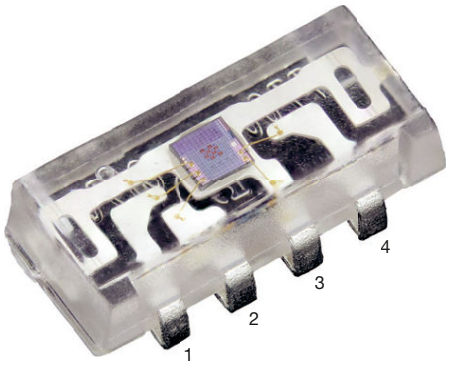
Vs=4~5 V

REVISION HISTORY

rev.	description	date
1.0	initial release	08/15/2016
1.01	updated datasheet	07/27/2017
1.02	updated to be certified to EN 62368-1 safety standard	07/09/2019
1.03	brand update	02/07/2020

The revision history provided is for informational purposes only and is believed to be accurate.

High Accuracy Ambient Light Sensor With I²C Interface



Pinning

1: SCL

2: V_{DD}

3: GND

4: SDA

DESCRIPTION

VEML7700 is a high accuracy ambient light digital 16-bit resolution sensor in a miniature transparent 6.8 mm x 2.35 mm x 3.0 mm package. It includes a high sensitive photo diode, a low noise amplifier, a 16-bit A/D converter and supports an easy to use I²C bus communication interface.

The ambient light result is as digital value available.

FEATURES

- Package type: surface-mount
- Package: side view
- Dimensions (L x W x H in mm): 6.8 x 2.35 x 3.0
- Integrated modules: ambient light sensor (ALS)
- Supply voltage range V_{DD}: 2.5 V to 3.6 V
- Communication via I²C interface
- Floor life: 72 h, MSL 4, according to J-STD-020
- Low shut down current consumption: typ. 0.5 µA
- Material categorization: for definitions of compliance please see www.vishay.com/doc?99912

Pb-free

e4

RoHS COMPLIANT

HALOGEN FREE GREEN (5-2008)

AMBIENT LIGHT FUNCTION

- Filtron™ technology adaption: close to real human eye response
- O-Trim™ technology adoption: ALS output tolerance ≤ 10 %
- 16-bit dynamic range for ambient light detection from 0 lx to about 120 klx with resolution down to 0.0036 lx/ct, supports low transmittance (dark) lens design
- 100 Hz and 120 Hz flicker noise rejection
- Excellent temperature compensation
- High dynamic detection resolution
- Software shutdown mode control

APPLICATIONS

- Ambient light sensor for backlight dimming of e.g. TV displays, smart phones, touch phones, PDA, GPS
- Ambient light sensor for industrial on- / off-lighting operation
- Optical switch for consumer, computing, and industrial devices and displays

PRODUCT SUMMARY

PART NUMBER	OPERATING RANGE (mm)	OPERATING VOLTAGE RANGE (V)	I²C BUS VOLTAGE RANGE (V)	AMBIENT LIGHT RANGE (lx)	AMBIENT LIGHT RESOLUTION (lx)	OUTPUT CODE	ADC RESOLUTION PROXIMITY / AMBIENT LIGHT
VEML7700	n/a	2.5 to 3.6	1.7 to 3.6	0 to 120 000	0.0036	16 bit, I²C	- / 0.0036

ORDERING INFORMATION

ORDERING CODE	PACKAGING	VOLUME (1)	REMARKS
VEML7700-TR	Tape and reel	MOQ: 2300 (MOQ is one reel)	Side view
VEML7700-TT	Tape and reel	MOQ: 2200 (MOQ is one reel)	Top view

Note

(1) MOQ: minimum order quantity

ABSOLUTE MAXIMUM RATINGS (T_{amb} = 25 °C, unless otherwise specified)					
PARAMETER	**TEST CONDITION**	**SYMBOL**	**MIN.**	**MAX.**	**UNIT**
Supply voltage		V_{DD}	0	4	V
Operation temperature range		T_{amb}	-25	+85	°C
Storage temperature range		T_{stg}	-25	+85	°C
Total power dissipation	$T_{amb} \leq 25$ °C	P_{tot}	-	50	mW
Junction temperature		T_j	-	100	°C

BASIC CHARACTERISTICS (T_{amb} = 25 °C, unless otherwise specified)						
PARAMETER	**TEST CONDITION**	**SYMBOL**	**MIN.**	**TYP.**	**MAX.**	**UNIT**
Supply voltage		V_{DD}	2.5	3.3	3.6	V
Shut down current (rem_2)	V_{DD} is 3.3 V	I_{sd}	-	0.5	-	µA
Operation mode current (rem_2)	V_{DD} is 3.3 V, PSM = 11, refresh time 4100 ms	I_{DD}	-	2	-	µA
	V_{DD} is 3.3 V, PSM = 00, refresh time 600 ms	I_{DD}	-	8	-	µA
	V_{DD} is 3.3 V, PSM_EN = 0, refresh time 100 ms	I_{DD}		45	-	µA
I^2C clock rate range		f_{SCL}	10	-	400	kHz
I^2C bus input H-level range	V_{DD} is 3.3 V	V_{ih}	1.3	-	3.6	V
I^2C bus input L-level range	V_{DD} is 3.3 V	V_{il}	-	-	0.4	V
Digital current out (low, current sink)		I_{ol}	3	-	-	mA
Digital resolution (LSB count)	With ALS_GAIN = “01”		-	0.0036	-	lx/step
Detectable minimum illuminance	With ALS_GAIN = “01”	$E_{V\ min.}$	-	0.0072	-	lx
Detectable maximum illuminance	With ALS_GAIN = “10”	$E_{V\ max.}$	-	120 000	-	lx
Dark offset (rem_2)	With ALS_GAIN = “01”		-	3	-	step

Note

- rem_1: light source: white LED
  rem_2: light conditions: dark

CIRCUIT BLOCK DIAGRAM

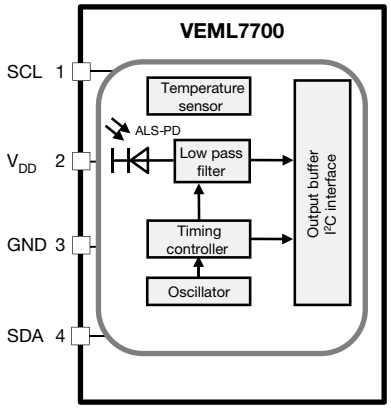


Fig. 1 - Block Diagram

BASIC CHARACTERISTICS (T_{amb} = 25 °C, unless otherwise specified)

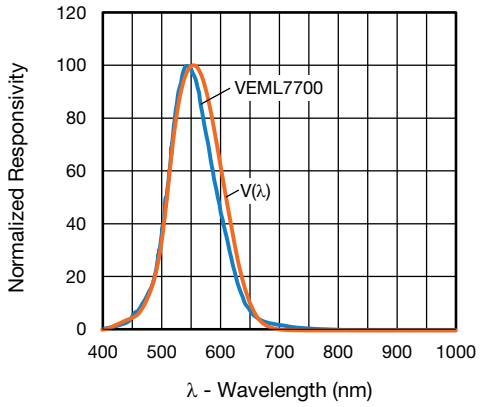


Fig. 2 - Spectral Response

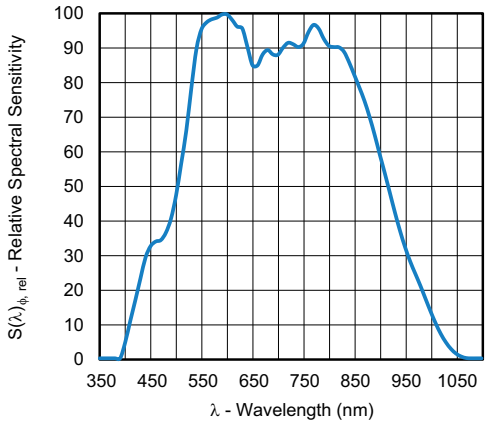


Fig. 3 - White Channel Sensitivity Spectrum

ALS sensitivity spectrum close to human eye photopic curve v(λ). Human eye curve adaption achieved by Filtron™ technology.

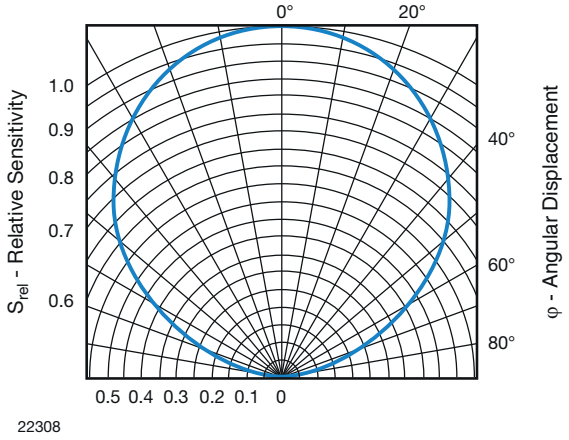


Fig. 4 - Relative Radiant Sensitivity vs. Angular Displacement

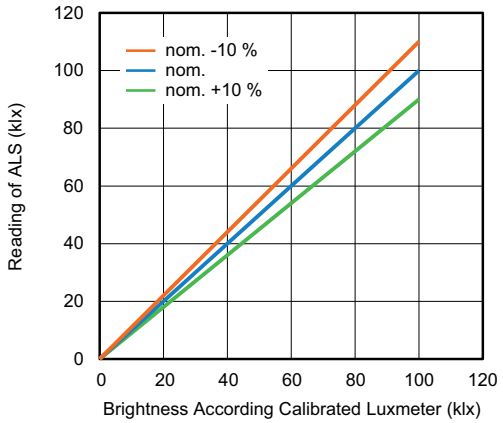


Fig. 5 - ALS measurement deviation between different light sources: ≤ 10 %

APPLICATION INFORMATION

VEML7700 is a cost effective solution of ambient light sensor with I²C bus interface. The standard serial digital interface is easy to access “Ambient Light Signal” without complex calculation and programming by external controller.

1. Application Circuit

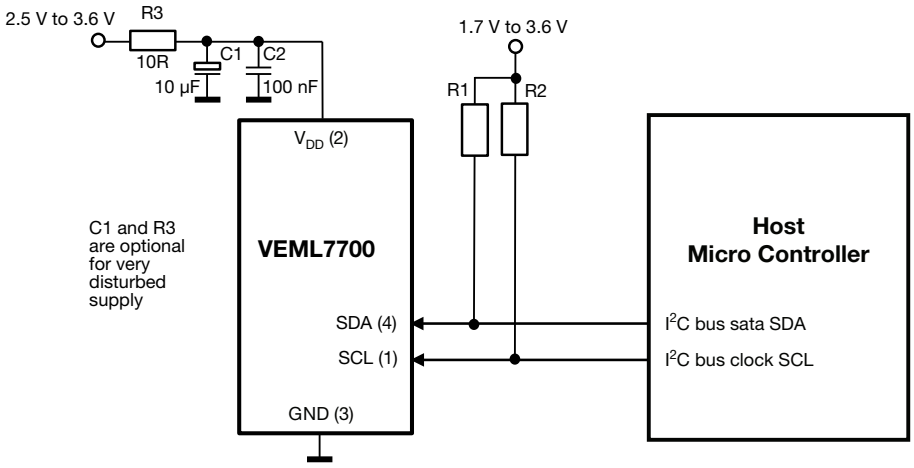


Fig. 6 - Application Diagram

Notes

- Proposed values for the pull-up resistor R1 and R2 should be > 1 kΩ, e.g. 2.2 kΩ to 4.7 kΩ.

  For detailed description about set-up and use as well as more application related information see AN: “Designing VEML7700 into an Application”

2. I²C Interface

The VEML7700 contains actual six 16 bit command codes for operation control, parameter setup, and result buffering. All registers are accessible via I²C communication. Figure 7 shows the basic I²C communication with VEML7700.

The built in I²C interface is compatible with I²C modes “standard” and “fast”: 10 kHz to 400 kHz.

I²C H-level range = 1.3 V to 3.6 V.

Please refer to the I²C specification from NXP for details.

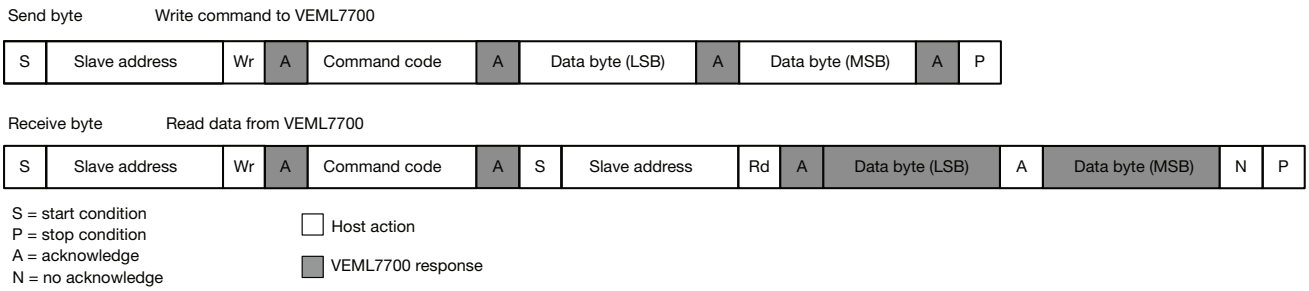


Fig. 7 - Send Byte / Receive Byte Protocol

Register Addresses

VEML7700 has actual six user accessible 16 bit command codes.

The addresses are 00h to 06h (03h not defined / reserved).

Device Address

The VEML7700 has a fix slave address for the host programming and accessing selection.

The slave address (7 bit) is set to 0010000 = 0x10.

The least significant bit (LSB) defines read or write mode.

According 8 bit the bus address is then 0010 0000 = 20h for write and 0010 0001 = 21h for read.

Auto-Memorization

VEML7700 can memorize the last ambient data before shutdown and keep this data before waking up.

When VEML7700 is in shutdown mode, the host can freely read this data via read command directly.

When VEML7700 wakes up, the data will be refreshed by new detection.

Interrupt pin not available for VEML7700

COMMAND REGISTER FORMAT				
COMMAND CODE	REGISTER NAME	BIT	FUNCTION / DESCRIPTION	R / W
00	ALS_CONF_0	15 : 0	ALS gain, integration time, interrupt, and shutdown	W
01	ALS_WH	15 : 8	ALS high threshold window setting (MSB)	W
		7 : 0	ALS high threshold window setting (LSB)	W
02	ALS_WL	15 : 8	ALS low threshold window setting (MSB)	W
		7 : 0	ALS low threshold window setting (LSB)	W
03	Power saving	15 : 0	Set (15 : 3) 0000 0000 0000 0b	
04	ALS	15 : 8	MSB 8 bits data of whole ALS 16 bits	R
		7 : 0	LSB 8 bits data of whole ALS 16 bits	R
05	WHITE	15 : 8	MSB 8 bits data of whole WHITE 16 bits	R
		7 : 0	LSB 8 bits data of whole WHITE 16 bits	R
06	ALS_INT	15 : 0	ALS INT trigger event	R

Note

- Command code 0 default value is 01 = devices is shut down

Command Code #0: Configuration Register

Register address = 00h
The command code #0 is for configuration of the ambient light measurements.

TABLE 1 - CONFIGURATION REGISTER #0			
REGISTER NAME	BIT	FUNCTION / DESCRIPTION	R / W
Reserved	15 : 13	Set 000b	W
ALS_GAIN	12 : 11	Gain selection 00 = ALS gain x 1 01 = ALS gain x 2 10 = ALS gain x (1/8) 11 = ALS gain x (1/4)	W
reserved	10	Set 0b	W
ALS_IT	9 : 6	ALS integration time setting 1100 = 25 ms 1000 = 50 ms 0000 = 100 ms 0001 = 200 ms 0010 = 400 ms 0011 = 800 ms	W
ALS_PERS	5 : 4	ALS persistence protect number setting 00 = 1 01 = 2 10 = 4 11 = 8	W
Reserved	3 : 2	Set 00b	W
ALS_INT_EN	1	ALS interrupt enable setting 0 = ALS INT disable 1 = ALS INT enable	W
ALS_SD	0	ALS shut down setting 0 = ALS power on 1 = ALS shut down	W

Note

- Light level [lx] is (ALS OUTPUT DATA [dec.] / ALS Gain x responsivity). Please study also the application note

Command Code #1: High Threshold Windows Setting

Command code address = 01h. Once enable INT function and use high / low windows threshold, bit 15:0 provides 16 bit register for high bound threshold windows setting.

TABLE 2 - HIGH THRESHOLD WINDOWS SETTING #1

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Description															
ALS high threshold window setting (15:8 MSB 8 bits of whole 16 bits)															
ALS high threshold window setting (7:0 LSB 8 bits of whole 16 bits)															

Command Code #2: Low Threshold Windows Setting

Command code address = 02h. Once enable INT function and use high / low windows threshold, bit 15:0 provides 16 bit register for low bound threshold windows setting.

TABLE 3 - LOW THRESHOLD WINDOWS SETTING #2

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Description															
ALS low threshold window setting (15:8 MSB 8 bits of whole 16 bits)															
ALS low threshold window setting (7:0 LSB 8 bits of whole 16 bits)															

Command Code #3: Power Saving Mode: PSM

Command code address = 03h. Bits 2 and 1 define the power saving modes. Bits 15 : 3 are reserved.

TABLE 4 - POWER SAVING MODES

REGISTER NAME	BIT	FUNCTION / DESCRIPTION	R / W
PSM	2 : 1	Power saving mode; see table “Refresh time” 00 = mode 1 01 = mode 2 10 = mode 3 11 = mode 4	W
PSM_EN	0	Power saving mode enable setting 0 = disable 1 = enable	W

Command Code #4: ALS High Resolution Output Data

Command code address = 04h. To access 16 bit high resolution ALS output, it is suitable to follow read protocol to read from command code 04 16 bits register.

TABLE 5 - ALS HIGH RESOLUTION OUTPUT DATA #4

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Description															
ALS high resolution output data (15 : 8 MSB 8 bits of whole 16 bits)															
ALS high resolution output data (7 : 0 LSB 8 bits of whole 16 bits)															

Command Code #5: White Channel Output Data

Command code address = 05h. To access 16 bit WHITE output, it is suitable to follow read protocol to read from command code 05 16 bits register.

TABLE 6 - WHITE CHANNEL OUTPUT DATA #5

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Description															
WHITE output data (15 : 8 MSB 8 bits of whole 16 bits)															
WHITE output data (7 : 0 LSB 8 bits of whole 16 bits)															

Command Code #6: Interrupt Status

Command code address = 06h. Bit 15 defines interrupt flag while trigger occurred due to data crossing low threshold windows. Bit 14 defines interrupt flag while trigger occurred due to data crossing high threshold windows.

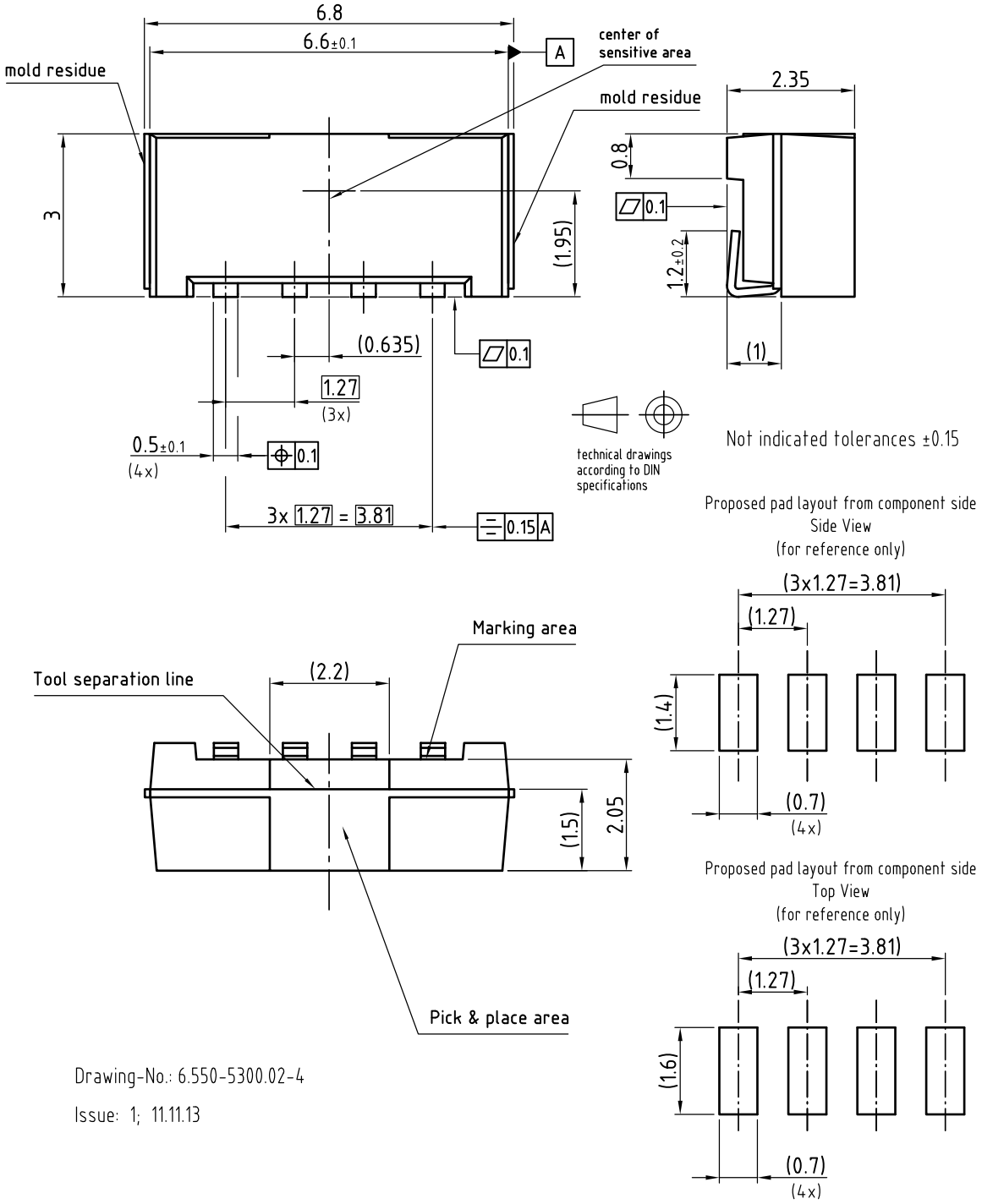
TABLE 7 - INTERRUPT STATUS #6		
Bit 15	Bit 14	Bit 13 to 0
int_th_low	int_th_high	reserved
Description		
int_th_low	R bit. Indicated a low threshold exceed	
int_th_high	R bit. Indicated a high threshold exceed	

REFRESH TIME DETERMINATION OF PSM

VEML7700's refresh time can be determined by PSM and ALS_IT setting in power saving mode (PSM). Cooperating with the command register setting, the designer has a flexible method in defining the timing, power consumption, and sensitivity for light data collection.

REFRESH TIME, I_{DD}, AND RESOLUTION RELATION					
ALS_GAIN	PSM	ALS_IT	REFRESH TIME (ms)	I_{DD} (µA)	RESOLUTION (lx/bit)
01	00	0000	600	8	0.0288
01	01	0000	1100	5	0.0288
01	10	0000	2100	3	0.0288
01	11	0000	4100	2	0.0288
01	00	0001	700	13	0.0144
01	01	0001	1200	8	0.0144
01	10	0001	2200	5	0.0144
01	11	0001	4200	3	0.0144
01	00	0010	900	20	0.0072
01	01	0010	1400	13	0.0072
01	10	0010	2400	8	0.0072
01	11	0010	4400	5	0.0072
01	00	0011	1300	28	0.0036
01	01	0011	1800	20	0.0036
01	10	0011	2800	13	0.0036
01	11	0011	4800	8	0.0036

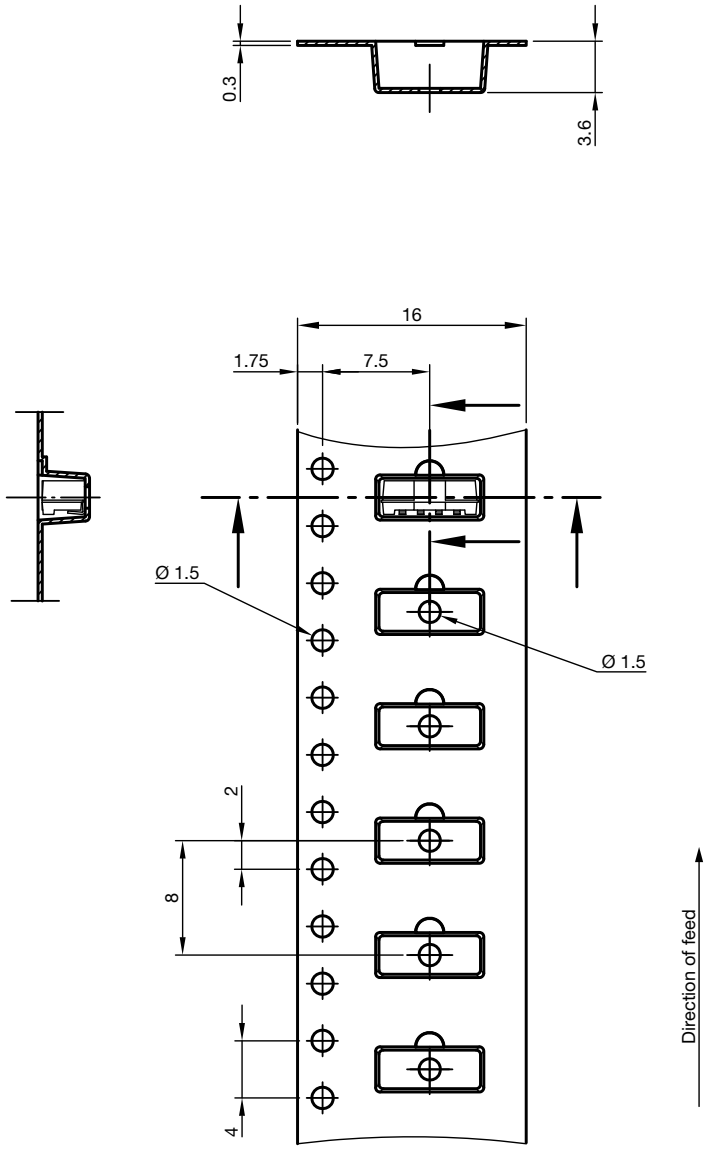
PACKAGE DIMENSIONS in millimeters



Drawing-No.: 6.550-5300.02-4

Issue: 1; 11.11.13

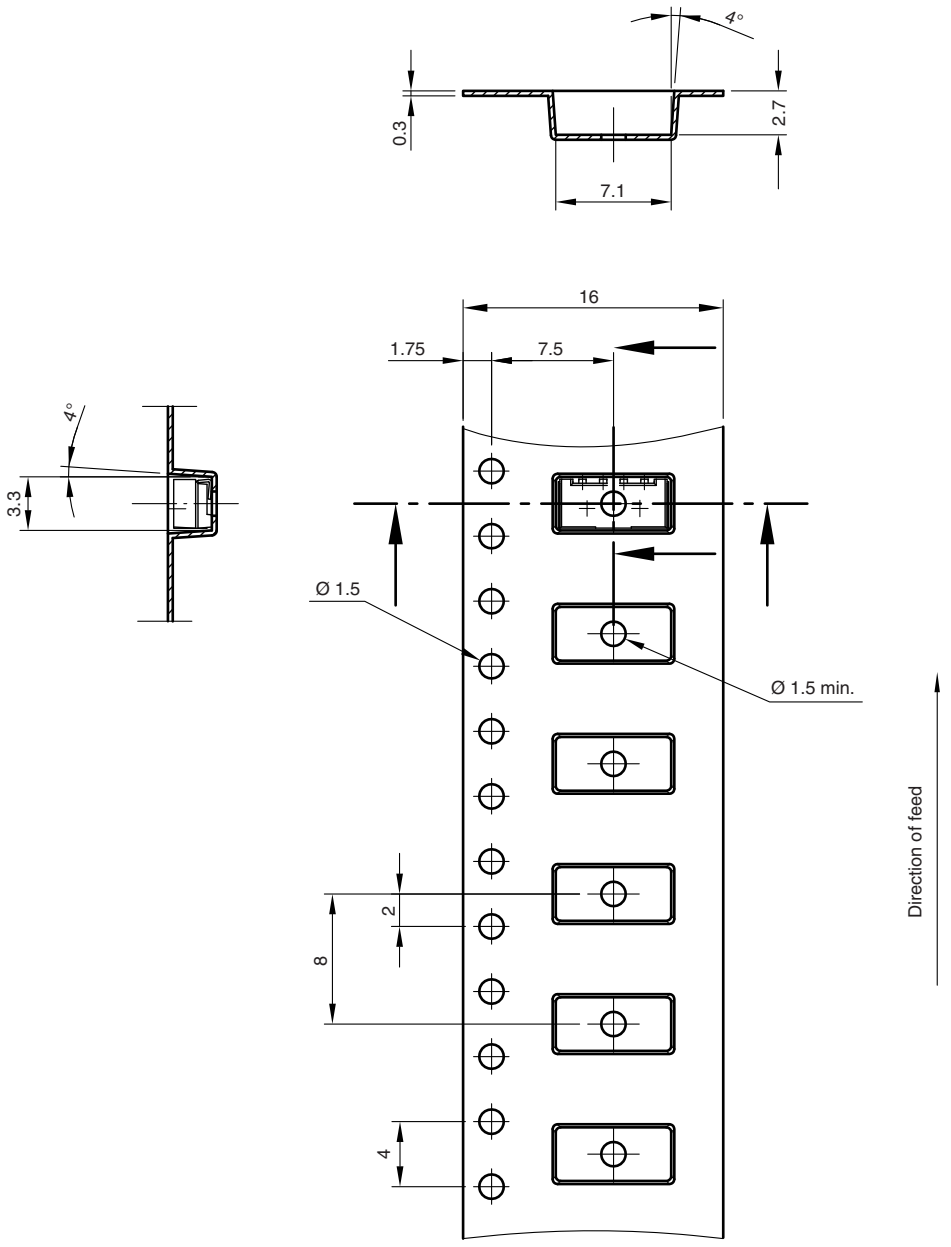
TAPING SIDE VIEW (-TR VERSION) in millimeters



Drawing-No.: 9.700-5342.01-4
Issue: 2; 12.06.13

technical drawings
according to DIN
specifications

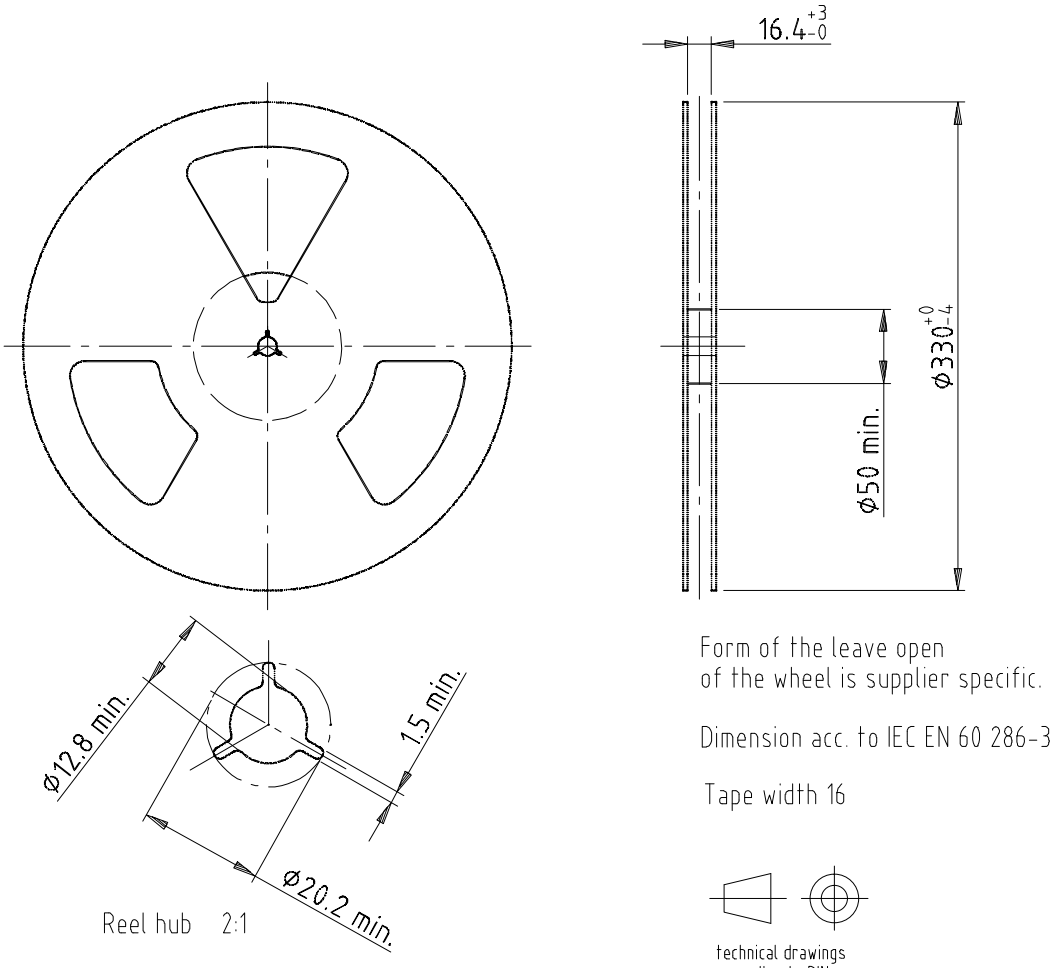
TAPING TOP VIEW (-TT VERSION) in millimeters



technical drawings
according to DIN
specifications

Drawing-No.: 9.700-5341.01-4
Issue: 2: 23.03.09
21666

REEL DIMENSIONS in millimeters



Form of the leave open
of the wheel is supplier specific.

Dimension acc. to IEC EN 60 286-3

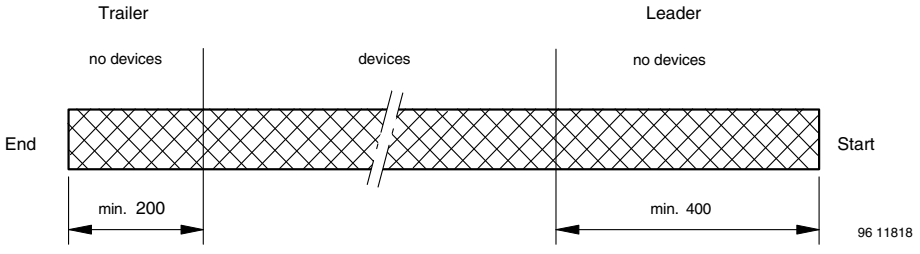
Tape width 16

technical drawings
according to DIN
specifications

Drawing-No.: 9.800-5052.V2-4
Issue: 1; 07.05.02

16734

LEADER AND TRAILER DIMENSIONS in millimeters



REFLOW SOLDER PROFILE

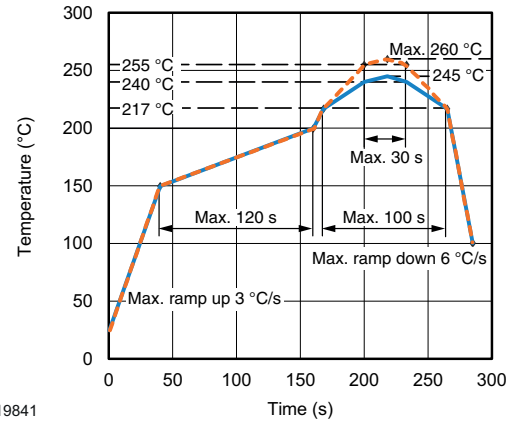


Fig. 8 - Lead (Pb)-free Reflow Solder Profile According to J-STD-020

DRYPACK

Devices are packed in moisture barrier bags (MBB) to prevent the products from moisture absorption during transportation and storage. Each bag contains a desiccant.

FLOOR LIFE

Floor life (time between soldering and removing from MBB) must not exceed the time indicated on MBB label:

Floor life: 72 h

Conditions: T_{amb} < 30 °C, RH < 60 %

Moisture sensitivity level 4, according to J-STD-020.

DRYING

In case of moisture absorption devices should be baked before soldering. Conditions see J-STD-020 or label. Devices taped on reel dry using recommended conditions 192 h at 40 °C (+ 5 °C), RH < 5 %.

Disclaimer

ALL PRODUCT, PRODUCT SPECIFICATIONS AND DATA ARE SUBJECT TO CHANGE WITHOUT NOTICE TO IMPROVE RELIABILITY, FUNCTION OR DESIGN OR OTHERWISE.

Vishay Intertechnology, Inc., its affiliates, agents, and employees, and all persons acting on its or their behalf (collectively, “Vishay”), disclaim any and all liability for any errors, inaccuracies or incompleteness contained in any datasheet or in any other disclosure relating to any product.

Vishay makes no warranty, representation or guarantee regarding the suitability of the products for any particular purpose or the continuing production of any product. To the maximum extent permitted by applicable law, Vishay disclaims (i) any and all liability arising out of the application or use of any product, (ii) any and all liability, including without limitation special, consequential or incidental damages, and (iii) any and all implied warranties, including warranties of fitness for particular purpose, non-infringement and merchantability.

Statements regarding the suitability of products for certain types of applications are based on Vishay’s knowledge of typical requirements that are often placed on Vishay products in generic applications. Such statements are not binding statements about the suitability of products for a particular application. It is the customer’s responsibility to validate that a particular product with the properties described in the product specification is suitable for use in a particular application. Parameters provided in datasheets and / or specifications may vary in different applications and performance may vary over time. All operating parameters, including typical parameters, must be validated for each customer application by the customer’s technical experts. Product specifications do not expand or otherwise modify Vishay’s terms and conditions of purchase, including but not limited to the warranty expressed therein.

Except as expressly indicated in writing, Vishay products are not designed for use in medical, life-saving, or life-sustaining applications or for any other application in which the failure of the Vishay product could result in personal injury or death. Customers using or selling Vishay products not expressly indicated for use in such applications do so at their own risk. Please contact authorized Vishay personnel to obtain written terms and conditions regarding products designed for such applications.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document or by any conduct of Vishay. Product names and markings noted herein may be trademarks of their respective owners.

Si7021-A20

I²C Humidity and Temperature Sensor

Features

- Precision Relative Humidity Sensor
  - ± 3% RH (max), 0–80% RH
- High Accuracy Temperature Sensor
  - ±0.4 °C (max), –10 to 85 °C
- 0 to 100% RH operating range
- Up to –40 to +125 °C operating range
- Wide operating voltage (1.9 to 3.6 V)
- Low Power Consumption
  - 150 µA active current
  - 60 nA standby current
- Factory-calibrated
- I²C Interface
- Integrated on-chip heater
- 3x3 mm DFN Package
- Excellent long term stability
- Optional factory-installed cover
  - Low-profile
  - Protection during reflow
  - Excludes liquids and particulates

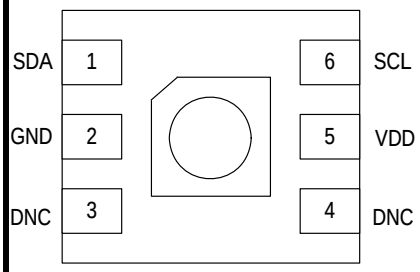
Ordering Information:
See page 29.

Applications

- HVAC/R
- Thermostats/humidistats
- Respiratory therapy
- White goods
- Indoor weather stations
- Micro-environments/data centers
- Automotive climate control and defogging
- Asset and goods tracking
- Mobile phones and tablets

Pin Assignments

Top View



Patent Protected. Patents pending

Description

The Si7021 I²C Humidity and Temperature Sensor is a monolithic CMOS IC integrating humidity and temperature sensor elements, an analog-to-digital converter, signal processing, calibration data, and an I²C Interface. The patented use of industry-standard, low-K polymeric dielectrics for sensing humidity enables the construction of low-power, monolithic CMOS Sensor ICs with low drift and hysteresis, and excellent long term stability.

The humidity and temperature sensors are factory-calibrated and the calibration data is stored in the on-chip non-volatile memory. This ensures that the sensors are fully interchangeable, with no recalibration or software changes required.

The Si7021 is available in a 3x3 mm DFN package and is reflow solderable. It can be used as a hardware- and software-compatible drop-in upgrade for existing RH/temperature sensors in 3x3 mm DFN-6 packages, featuring precision sensing over a wider range and lower power consumption. The optional factory-installed cover offers a low profile, convenient means of protecting the sensor during assembly (e.g., reflow soldering) and throughout the life of the product, excluding liquids (hydrophobic/oleophobic) and particulates.

The Si7021 offers an accurate, low-power, factory-calibrated digital solution ideal for measuring humidity, dew-point, and temperature, in applications ranging from HVAC/R and asset tracking to industrial and consumer platforms.

Functional Block Diagram

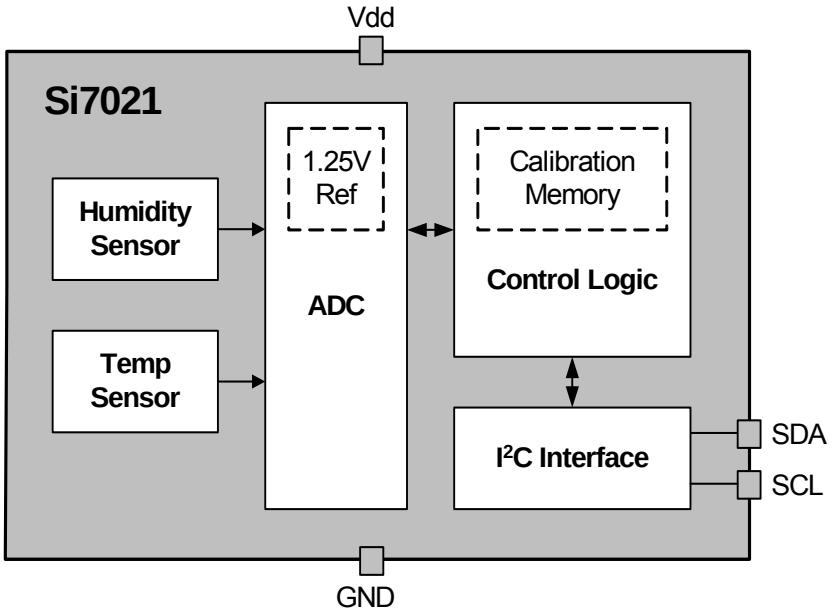


TABLE OF CONTENTS

1. Electrical Specifications

Unless otherwise specified, all min/max specifications apply over the recommended operating conditions.

Table 1. Recommended Operating Conditions

Parameter	Symbol	Test Condition	Min	Typ	Max	Unit
Power Supply	V_{DD}		1.9	—	3.6	V
Operating Temperature	T_A	I and Y grade	–40	—	+125	°C
Operating Temperature	T_A	G grade	–40	—	+85	°C

Table 2. General Specifications

1.9 ≤ V_{DD} ≤ 3.6 V; T_A = –40 to 85 °C (G grade) or –40 to 125 °C (I/Y grade); default conversion time unless otherwise noted.

Parameter	Symbol	Test Condition	Min	Typ	Max	Unit
Input Voltage High	V_{IH}	SCL, SDA pins	0.7xV_{DD}	—	—	V
Input Voltage Low	V_{IL}	SCL, SDA pins	—	—	0.3xV_{DD}	V
Input Voltage Range	V_{IN}	SCL, SDA pins with respect to GND	0.0	—	V_{DD}	V
Input Leakage	I_{IL}	SCL, SDA pins	—	—	1	µA
Output Voltage Low	V_{OL}	SDA pin; I_{OL} = 2.5 mA; V_{DD} = 3.3 V	—	—	0.6	V
		SDA pin; I_{OL} = 1.2 mA; V_{DD} = 1.9 V	—	—	0.4	V
Current Consumption	I_{DD}	RH conversion in progress	—	150	180	µA
		Temperature conversion in progress	—	90	120	µA
		Standby, –40 to +85 °C[2]	—	0.06	0.62	µA
		Standby, –40 to +125 °C[2]	—	0.06	3.8	µA
		Peak I_{DD} during powerup[3]	—	3.5	4.0	mA
		Peak I_{DD} during I^2C operations[4]	—	3.5	4.0	mA
Heater Current[5]	I_{HEAT}		—	3.1 to 94.2	—	mA

Notes:

1. Initiating a RH measurement will also automatically initiate a temperature measurement. The total conversion time will be t_{CONV}(RH) + t_{CONV}(T).
2. No conversion or I^2C transaction in progress. Typical values measured at 25 °C.
3. Occurs once during powerup. Duration is <5 msec.
4. Occurs during I^2C commands for Reset, Read/Write User Registers, Read EID, and Read Firmware Version. Duration is <100 µs when I^2C clock speed is >100 kHz (>200 kHz for 2-byte commands).
5. Additional current consumption when HTRE bit enabled. See Section “5.5. Heater” for more information.

Table 2. General Specifications (Continued)

1.9 ≤ VDD ≤ 3.6 V; TA = –40 to 85 °C (G grade) or –40 to 125 °C (I/Y grade); default conversion time unless otherwise noted.

Parameter	Symbol	Test Condition	Min	Typ	Max	Unit
Conversion Time[1]	t_{CONV}	12-bit RH	—	10	12	ms
		11-bit RH	—	5.8	7	
		10-bit RH	—	3.7	4.5	
		8-bit RH	—	2.6	3.1	
		14-bit temperature	—	7	10.8	
		13-bit temperature	—	4	6.2	
		12-bit temperature	—	2.4	3.8	
		11-bit temperature	—	1.5	2.4	
Powerup Time	t_{PU}	From VDD ≥ 1.9 V to ready for a conversion, 25 °C	—	18	25	ms
		From VDD ≥ 1.9 V to ready for a conversion, full temperature range	—	—	80	
		After issuing a software reset command	—	5	15	

Notes:
1. Initiating a RH measurement will also automatically initiate a temperature measurement. The total conversion time will be t_{CONV}(RH) + t_{CONV}(T).
2. No conversion or I²C transaction in progress. Typical values measured at 25 °C.
3. Occurs once during powerup. Duration is <5 msec.
4. Occurs during I²C commands for Reset, Read/Write User Registers, Read EID, and Read Firmware Version. Duration is <100 µs when I²C clock speed is >100 kHz (>200 kHz for 2-byte commands).
5. Additional current consumption when HTRE bit enabled. See Section “5.5. Heater” for more information.

Table 3. I²C Interface Specifications[1]

1.9 ≤ V_{DD} ≤ 3.6 V; T_A = –40 to +85 °C (G grade) or –40 to +125 °C (I/Y grade) unless otherwise noted.

Parameter	Symbol	Test Condition	Min	Typ	Max	Unit
Hysteresis	V_{HYS}	High-to-low versus low-to-high transition	0.05 x V_{DD}	—	—	V
SCLK Frequency[2]	f_{SCL}		—	—	400	kHz
SCL High Time	t_{SKH}		0.6	—	—	µs
SCL Low Time	t_{SKL}		1.3	—	—	µs
Start Hold Time	t_{STH}		0.6	—	—	µs
Start Setup Time	t_{STS}		0.6	—	—	µs

Notes:
1. All values are referenced to V_{IL} and/or V_{IH}.
2. Depending on the conversion command, the Si7021 may hold the master during the conversion (clock stretch). At above 100 kHz SCL, the Si7021 may also hold the master briefly for user register and device ID transactions. At the highest I²C speed of 400 kHz the stretching will be <50 µs.
3. Pulses up to and including 50 ns will be suppressed.

Table 3. I²C Interface Specifications[1] (Continued)

1.9 ≤ V_{DD} ≤ 3.6 V; T_A = –40 to +85 °C (G grade) or –40 to +125 °C (I/Y grade) unless otherwise noted.

Parameter	Symbol	Test Condition	Min	Typ	Max	Unit
Stop Setup Time	t_{SPS}		0.6	—	—	µs
Bus Free Time	t_{BUF}	Between Stop and Start	1.3	—	—	µs
SDA Setup Time	t_{DS}		100	—	—	ns
SDA Hold Time	t_{DH}		100	—	—	ns
SDA Valid Time	$t_{VD;DAT}$	From SCL low to data valid	—	—	0.9	µs
SDA Acknowledge Valid Time	$t_{VD;ACK}$	From SCL low to data valid	—	—	0.9	µs
Suppressed Pulse Width[3]	t_{SPS}		50	—	—	ns

Notes:
1. All values are referenced to V_{IL} and/or V_{IH}.
2. Depending on the conversion command, the Si7021 may hold the master during the conversion (clock stretch). At above 100 kHz SCL, the Si7021 may also hold the master briefly for user register and device ID transactions. At the highest I²C speed of 400 kHz the stretching will be <50 µs.
3. Pulses up to and including 50 ns will be suppressed.

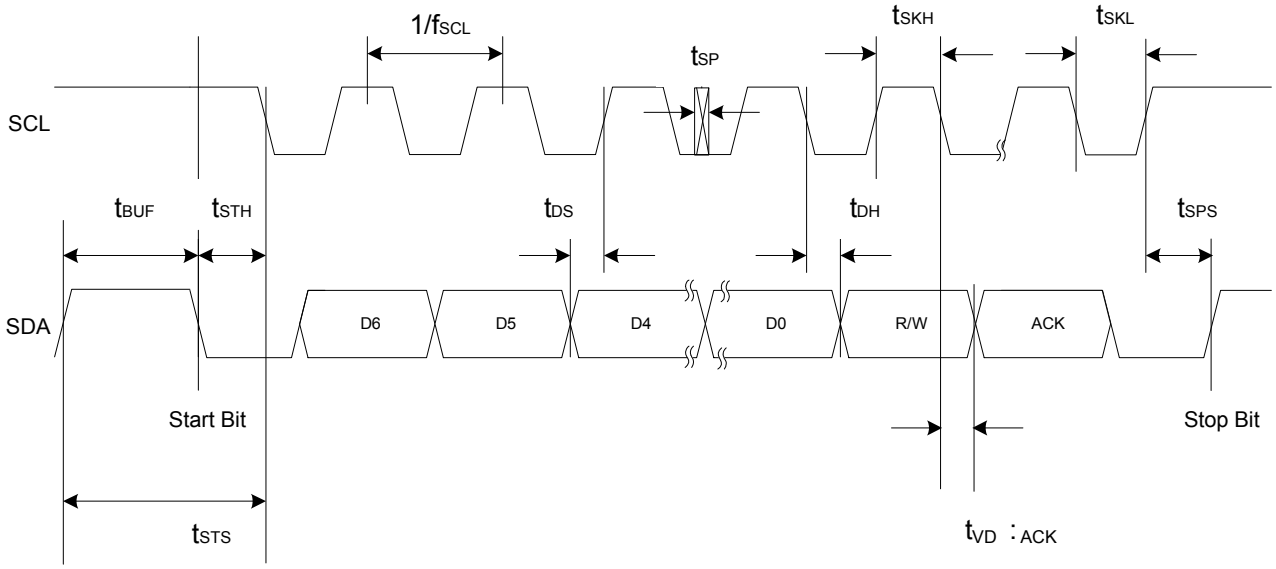


Figure 1. I²C Interface Timing Diagram

Table 4. Humidity Sensor

1.9 ≤ V_{DD} ≤ 3.6 V; T_A = 30 °C; default conversion time unless otherwise noted.

Parameter	Symbol	Test Condition	Min	Typ	Max	Unit
Operating Range[1]		Non-condensing	0	—	100	%RH
Accuracy[2, 3]		0 – 80% RH	—	±2	±3	%RH
		80 – 100% RH	See Figure 2.			
Repeatability/Noise		12-bit resolution	—	0.025	—	%RH RMS
		11-bit resolution	—	0.05	—	
		10-bit resolution	—	0.1	—	
		8-bit resolution	—	0.2	—	
Response Time[4]	$\tau_{63\%}$	1 m/s airflow, with cover	—	18	—	S
		1 m/s airflow, without cover	—	17	—	
Drift vs. Temperature			—	0.05	—	%RH/°C
Hysteresis			—	±1	—	%RH
Long Term Stability[3]			—	≤ 0.25	—	%RH/yr

Notes:

1. Recommended humidity operating range is 20% to 80% RH (non-condensing) over –10 °C to 60 °C. Prolonged operation beyond these ranges may result in a shift of sensor reading, with slow recovery time.
2. Excludes hysteresis, long term drift, and certain other factors and is applicable to non-condensing environments only. See Section “4.1. Relative Humidity Sensor Accuracy” for more details.
3. Drift due to aging effects at typical room conditions of 30 °C and 30% to 50% RH. May be impacted by dust, vaporized solvents or other contaminants, e.g., out-gassing tapes, adhesives, packaging materials, etc. See Section “4.7. Long Term Drift/Aging” .
4. Response time to a step change in RH. Time for the RH output to change by 63% of the total RH change.

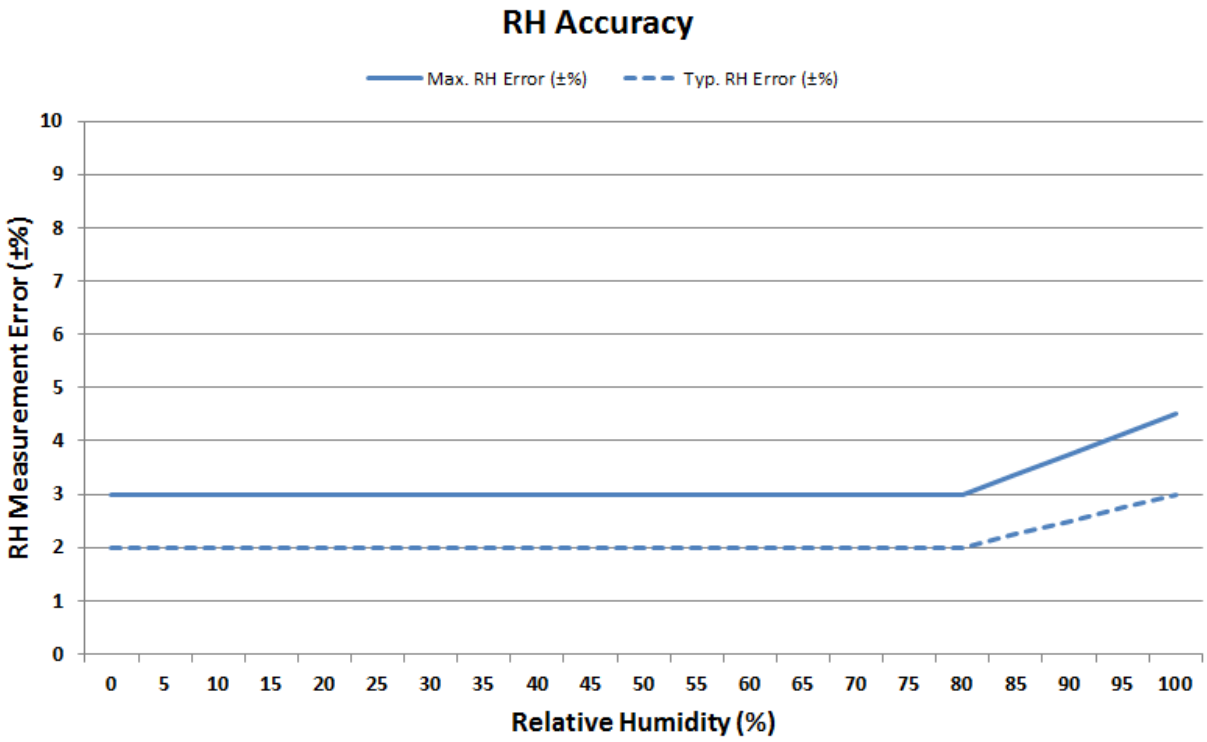


Figure 2. RH Accuracy at 30 °C

Table 5. Temperature Sensor

1.9 ≤ V_{DD} ≤ 3.6 V; T_A = –40 to +85 °C (G grade) or –40 to +125 °C (I/Y grade) default conversion time, unless otherwise noted.

Parameter	Symbol	Test Condition	Min	Typ	Max	Unit
Operating Range		I and Y Grade	–40	—	+125	°C
		G Grade	–40	—	+85	°C
Accuracy[1]		–10 °C≤ t_A ≤ 85 °C	—	±0.3	±0.4	°C
		–40 ≤ t_A ≤ 125 °C		Figure 3		
Repeatability/Noise		14-bit resolution	—	0.01	—	°C RMS
		13-bit resolution	—	0.02	—	
		12-bit resolution	—	0.04	—	
		11-bit resolution	—	0.08	—	
Response Time[2]	$T_{63\%}$	Unmounted device	—	0.7	—	s
		Si7021-EB board	—	5.1	—	
Long Term Stability			—	≤ 0.01	—	°C/Yr

Notes:
1. 14b measurement resolution (default).
2. Time to reach 63% of final value in response to a step change in temperature. Actual response time will vary dependent on system thermal mass and air-flow.

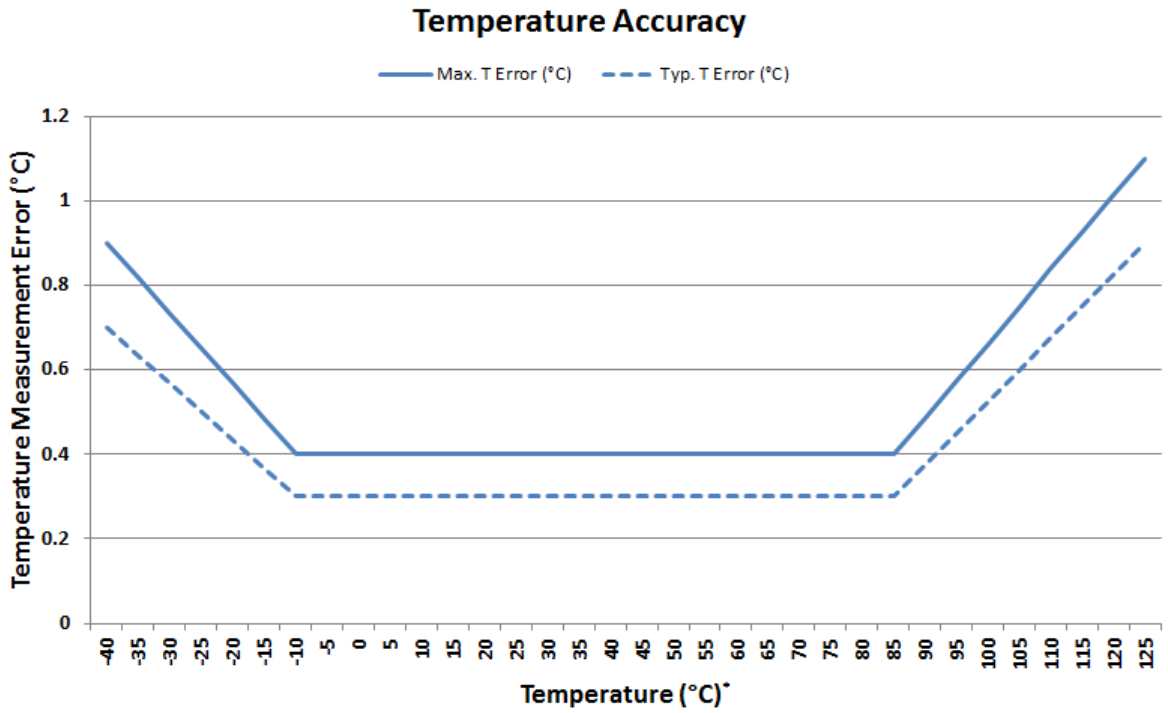


Figure 3. Temperature Accuracy*

***Note:** Applies only to I and Y grade devices beyond +85 °C.

Table 6. Thermal Characteristics

Parameter	Symbol	Test Condition	DFN-6	Unit
Junction to Air Thermal Resistance	θ_{JA}	JEDEC 2-Layer board, No Airflow	256	°C/W
Junction to Air Thermal Resistance	θ_{JA}	JEDEC 2-Layer board, 1 m/s Airflow	224	°C/W
Junction to Air Thermal Resistance	θ_{JA}	JEDEC 2-Layer board, 2.5 m/s Airflow	205	°C/W
Junction to Case Thermal Resistance	θ_{JC}	JEDEC 2-Layer board	22	°C/W
Junction to Board Thermal Resistance	θ_{JB}	JEDEC 2-Layer board	134	°C/W

Table 7. Absolute Maximum Ratings[1]

Parameter	Symbol	Test Condition	Min	Typ	Max	Unit
Ambient temperature under bias			–55	—	125	°C
Storage Temperature[2]			–65	—	150	°C
Voltage on I/O pins			–0.3	—	V_{DD}+0.3 V	V
Voltage on VDD with respect to GND			–0.3	—	4.2	V
ESD Tolerance		HBM	—	—	2	kV
		CDM	—	—	1.25	kV
		MM	—	—	250	V

Notes:
1. Absolute maximum ratings are stress ratings only, operation at or beyond these conditions is not implied and may shorten the life of the device or alter its performance.
2. Special handling considerations apply; see application note, “AN607: Si70xx Humidity Sensor Designer’s Guide”.

2. Typical Application Circuits

The primary function of the Si7021 is to measure relative humidity and temperature. Figure 4 demonstrates the typical application circuit to achieve these functions.

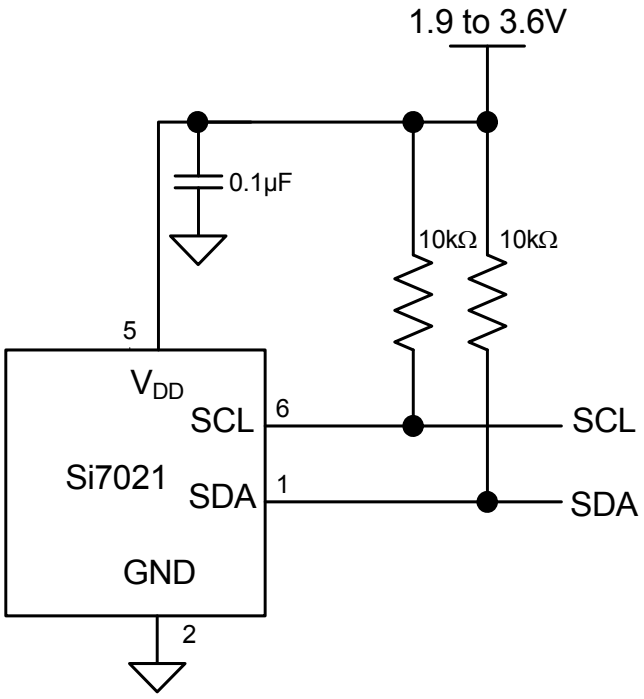


Figure 4. Typical Application Circuit for Relative Humidity and Temperature Measurement

3. Bill of Materials

Table 8. Typical Application Circuit BOM for Relative Humidity and Temperature Measurement

Reference	Description	Mfr Part Number	Manufacturer
R1	Resistor, 10 kΩ, ±5%, 1/16 W, 0603	CR0603-16W-103JT	Venkel
R2	Resistor, 10 kΩ, ±5%, 1/16 W, 0603	CR0603-16W-103JT	Venkel
C1	Capacitor, 0.1 µF, 16 V, X7R, 0603	C0603X7R160-104M	Venkel
U1	IC, Digital Temperature/humidity Sensor	Si7021-A20-GM	Silicon Labs

4. Functional Description

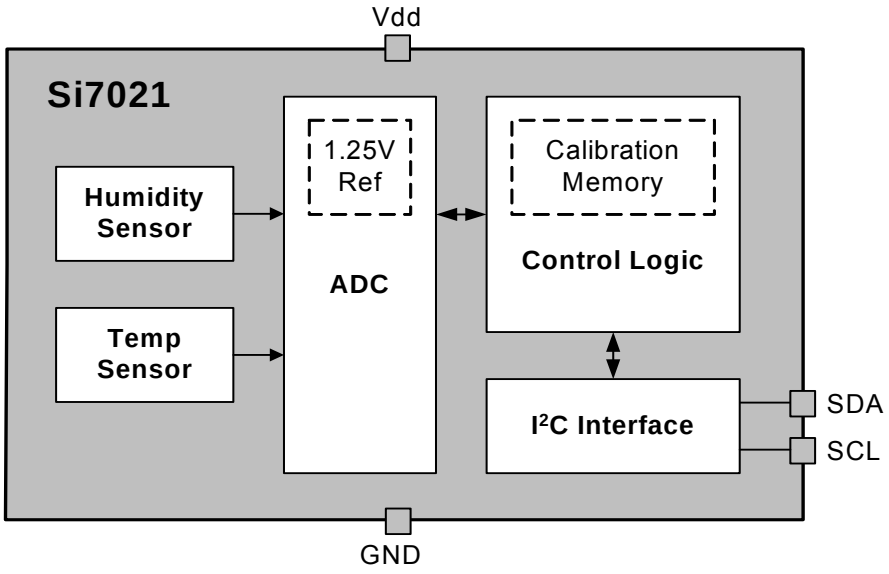


Figure 5. Si7021 Block Diagram

The Si7021 is a digital relative humidity and temperature sensor that integrates temperature and humidity sensor elements, an analog-to-digital converter, signal processing, calibration, polynomial non-linearity correction, and an I^2C interface all in a single chip. The Si7021 is individually factory-calibrated for both temperature and humidity, with the calibration data stored in on-chip non-volatile memory. This ensures that the sensor is fully interchangeable, with no recalibration or changes to software required. Patented use of industry-standard CMOS and low-K dielectrics as a sensor enables the Si7021 to achieve excellent long term stability and immunity to contaminants with low drift and hysteresis. The Si7021 offers a low-power, high-accuracy, calibrated and stable solution ideal for a wide range of temperature, humidity, and dew-point applications including medical and instrumentation, high-reliability automotive and industrial systems, and cost-sensitive consumer electronics.

While the Si7021 is largely a conventional mixed-signal CMOS integrated circuit, relative humidity sensors in general and those based on capacitive sensing using polymeric dielectrics have unique application and use requirements that are not common to conventional (non-sensor) ICs. Chief among those are:

- The need to protect the sensor during board assembly, i.e., solder reflow, and the need to subsequently rehydrate the sensor.
- The need to protect the senor from damage or contamination during the product life-cycle.
- The impact of prolonged exposure to extremes of temperature and/or humidity and their potential effect on sensor accuracy.
- The effects of humidity sensor “memory”.

Each of these items is discussed in more detail in the following sections.

4.1. Relative Humidity Sensor Accuracy

To determine the accuracy of a relative humidity sensor, it is placed in a temperature and humidity controlled chamber. The temperature is set to a convenient fixed value (typically 25–30 °C) and the relative humidity is swept from 20 to 80% and back to 20% in the following steps: 20% – 40% – 60% – 80% – 80% – 60% – 40% – 20%. At each set-point, the chamber is allowed to settle for a period of 60 minutes before a reading is taken from the sensor. Prior to the sweep, the device is allowed to stabilize to 50%RH. The solid trace in Figure 6, “Measuring Sensor Accuracy Including Hysteresis,” shows the result of a typical sweep.

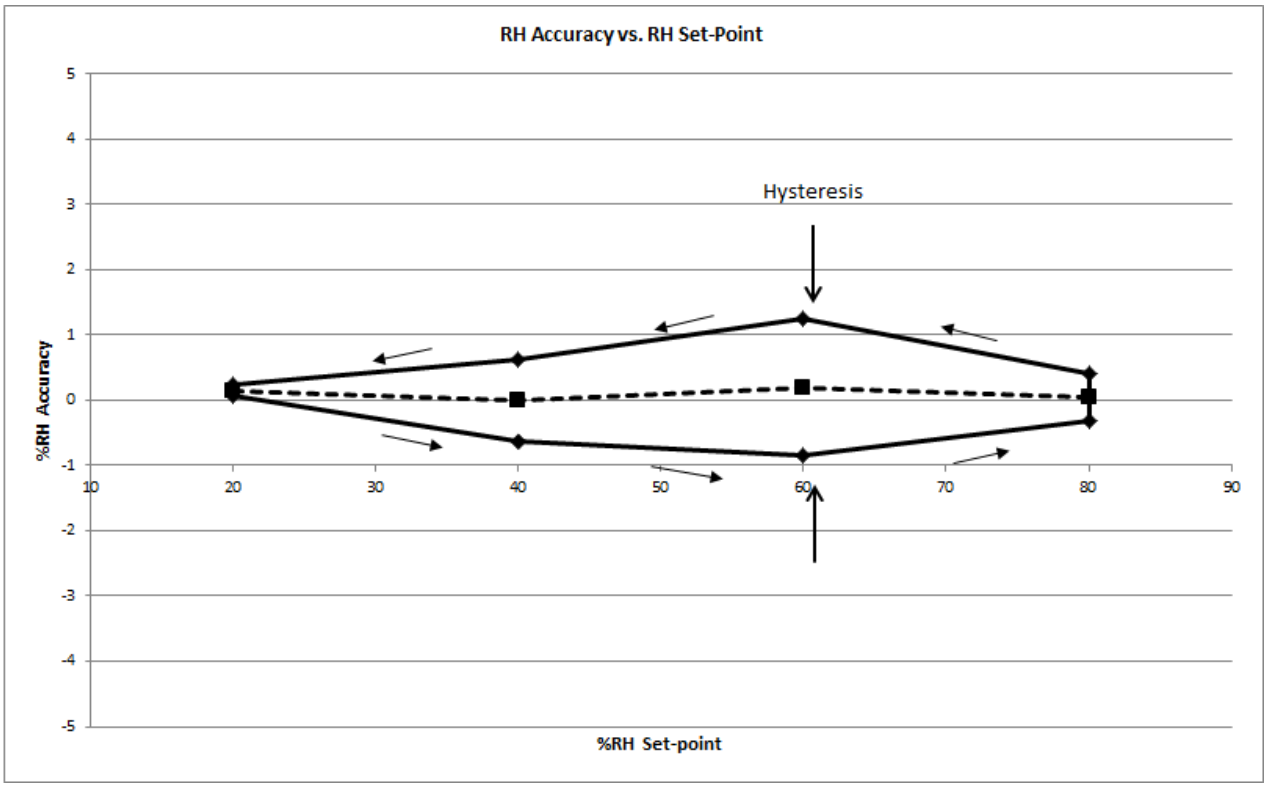


Figure 6. Measuring Sensor Accuracy Including Hysteresis

The RH accuracy is defined as the dotted line shown in Figure 6, which is the average of the two data points at each relative humidity set-point. In this case, the sensor shows an accuracy of 0.25%RH. The Si7021 accuracy specification (Table 4) includes:

- Unit-to-unit and lot-to-lot variation
- Accuracy of factory calibration
- Margin for shifts that can occur during solder reflow

The accuracy specification does not include:

- Hysteresis (typically ±1%)
- Effects from long term exposure to very humid conditions
- Contamination of the sensor by particulates, chemicals, etc.
- Other aging related shifts ("Long-term stability")
- Variations due to temperature (see Drift vs. Temperature in Table 4). RH readings will typically vary with temperature by less than ± 0.05% ° C.

4.2. Hysteresis

The moisture absorbent film (polymeric dielectric) of the humidity sensor will carry a memory of its exposure history, particularly its recent or extreme exposure history. A sensor exposed to relatively low humidity will carry a negative offset relative to the factory calibration, and a sensor exposed to relatively high humidity will carry a positive offset relative to the factory calibration. This factor causes a hysteresis effect illustrated by the solid trace in Figure 6. The hysteresis value is the difference in %RH between the maximum absolute error on the decreasing humidity ramp and the maximum absolute error on the increasing humidity ramp at a single relative humidity setpoint and is expressed as a bipolar quantity relative to the average error (dashed trace). In the example of Figure 6, the measurement uncertainty due to the hysteresis effect is ±1.0%RH.

4.3. Prolonged Exposure to High Humidity

Prolonged exposure to high humidity will result in a gradual upward drift of the RH reading. The shift in sensor reading resulting from this drift will generally disappear slowly under normal ambient conditions. The amount of shift is proportional to the magnitude of relative humidity and the length of exposure. In the case of lengthy exposure to high humidity, some of the resulting shift may persist indefinitely under typical conditions. It is generally possible to substantially reverse this affect by baking the device (see Section “4.6. Bake/Hydrate Procedure”).

4.4. PCB Assembly

4.4.1. Soldering

Like most ICs, Si7021 devices are shipped from the factory vacuum-packed with an enclosed desiccant to avoid any RH accuracy drift during storage and to prevent any moisture-related issues during solder reflow. The following guidelines should be observed during PCB assembly:

- Si7021 devices are compatible with standard board assembly processes. Devices should be soldered using reflow per the recommended card reflow profile. See Section “10. PCB Land Pattern and Solder Mask Design” for the recommended card reflow profile.
- A "no clean" solder process is recommended to minimize the need for water or solvent rinses after soldering. Cleaning after soldering is possible, but must be done carefully to avoid impacting the performance of the sensor. See “AN607: Si70xx Humidity Sensor Designer’s Guide” for more information on cleaning.
- It is essential that the exposed polymer sensing film be kept clean and undamaged. This can be accomplished by careful handling and a clean, well-controlled assembly process. When in doubt or for extra protection, a heat-resistant, protective cover such as Kapton™ KPPD-1/8 polyimide tape can be installed during PCB assembly.

Si7021s may be ordered with a factory-fitted, solder-resistant protective cover. This cover provides protection during PCB assembly or rework but without the time and effort required to install and remove the Kapton tape. It can be left in place for the lifetime of the product, preventing liquids, dust or other contaminants from coming into contact with the polymer sensor film. See Section “8. Ordering Guide” for a list of ordering part numbers that include the cover.

4.4.2. Rehydration

The measured humidity value will generally shift slightly after solder reflow. A portion of this shift is permanent and is accounted for in the accuracy specifications in Table 4. After soldering, an Si7021 should be allowed to equilibrate under controlled RH conditions (room temperature, 45–55%RH) for at least 48 hours to eliminate the remainder of the shift and return the device to its specified accuracy performance.

4.4.3. Rework

To maintain the specified sensor performance, care must be taken during rework to minimize the exposure of the device to excessive heat and to avoid damage/contamination or a shift in the sensor reading due to liquids, solder flux, etc. Manual touch-up using a soldering iron is permissible under the following guidelines:

- The exposed polymer sensing film must be kept clean and undamaged. A protective cover is recommended during any rework operation (Kapton® tape or the factory installed cover).
- Flux must not be allowed to contaminate the sensor; liquid flux is not recommended even with a cover in place. Conventional lead-free solder with rosin core is acceptable for touch-up as long as a cover is in place during the rework.
- If possible, avoid water or solvent rinses after touch-up. Cleaning after soldering is possible, but must be done carefully to avoid impacting the performance of the sensor. See AN607 for more information on cleaning.
- Minimize the heating of the device. Soldering iron temperatures should not exceed 350 °C and the contact time per pin should not exceed five seconds.
- Hot air rework is not recommended. If a device must be replaced, remove the device by hot air and solder a new part in its place by reflow following the guidelines above.

***Note:** All trademarks are the property of their respective owners.

Figure 7. Si7021 with Factory-Installed Protective Cover

4.5. Protecting the Sensor

Because the sensor operates on the principal of measuring a change in capacitance, any changes to the dielectric constant of the polymer film will be detected as a change in relative humidity. Therefore, it is important to minimize the probability of contaminants coming into contact with the sensor. Dust and other particles as well as liquids can affect the RH reading. It is recommended that a cover is employed in the end system that blocks contaminants but allows water vapor to pass through. Depending on the needs of the application, this can be as simple as plastic or metallic gauze for basic protection against particulates or something more sophisticated such as a hydrophobic membrane providing up to IP67 compliant protection.

The Si7021 may be ordered with a factory-fitted, solder-resistant cover that can be left in place for the lifetime of the product. It is very low-profile, hydrophobic and oleophobic. See Section “8. Ordering Guide” for a list of ordering part numbers that include the cover. A dimensioned drawing of the IC with the cover is included in Section “9. Package Outline” . Other characteristics of the cover are listed in Table 9.

Table 9. Specifications of Protective Cover

Parameter	Value
Material	PTFE
Operating Temperature	–40 to 125 °C
Maximum Reflow Temperature	260 °C
IP Rating (per IEC 529)	IP67

4.6. Bake/Hydrate Procedure

After exposure to extremes of temperature and/or humidity for prolonged periods, the polymer sensor film can become either very dry or very wet, in each case the result is either high or low relative humidity readings. Under normal operating conditions, the induced error will diminish over time. From a very dry condition, such as after shipment and soldering, the error will diminish over a few days at typical controlled ambient conditions, e.g., 48 hours of 45 ≤ %RH ≤ 55. However, from a very wet condition, recovery may take significantly longer. To accelerate recovery from a wet condition, a bake and hydrate cycle can be implemented. This operation consists of the following steps:

- Baking the sensor at 125 °C for ≥ 12 hours
- Hydration at 30 °C in 75% RH for ≥ 10 hours

Following this cycle, the sensor will return to normal operation in typical ambient conditions after a few days.

4.7. Long Term Drift/Aging

Over long periods of time, the sensor readings may drift due to aging of the device. Standard accelerated life testing of the Si7021 has resulted in the specifications for long-term drift shown in Table 4 and Table 5. This contribution to the overall sensor accuracy accounts only for the long-term aging of the device in an otherwise benign operating environment and does not include the effects of damage, contamination, or exposure to extreme environmental conditions.

5. I²C Interface

The Si7021 communicates with the host controller over a digital I²C interface. The 7-bit base slave address is 0x40.

Table 10. I²C Slave Address Byte

A6	A5	A4	A3	A2	A1	A0	R/W
1	0	0	0	0	0	0	0

Master I²C devices communicate with the Si7021 using a command structure. The commands are listed in the I²C command table. Commands other than those documented below are undefined and should not be sent to the device.

Table 11. I²C Command Table

Command Description	Command Code
Measure Relative Humidity, Hold Master Mode	0xE5
Measure Relative Humidity, No Hold Master Mode	0xF5
Measure Temperature, Hold Master Mode	0xE3
Measure Temperature, No Hold Master Mode	0xF3
Read Temperature Value from Previous RH Measurement	0xE0
Reset	0xFE
Write RH/T User Register 1	0xE6
Read RH/T User Register 1	0xE7
Write Heater Control Register	0x51
Read Heater Control Register	0x11
Read Electronic ID 1st Byte	0xFA 0x0F
Read Electronic ID 2nd Byte	0xFC 0xC9
Read Firmware Revision	0x84 0xB8

5.1. Issuing a Measurement Command

The measurement commands instruct the Si7021 to perform one of two possible measurements; Relative Humidity or Temperature. The procedure to issue any one of these commands is identical. While the measurement is in progress, the option of either clock stretching (Hold Master Mode) or Not Acknowledging read requests (No Hold Master Mode) is available to indicate to the master that the measurement is in progress; the chosen command code determines which mode is used.

Optionally, a checksum byte can be returned from the slave for use in checking for transmission errors. The checksum byte will follow the least significant measurement byte if it is acknowledged by the master. The checksum byte is not returned if the master “not acknowledges” the least significant measurement byte. The checksum byte is calculated using a CRC generator polynomial of $x^8 + x^5 + x^4 + 1$, with an initialization of 0x00.

The checksum byte is optional after initiating an RH or temperature measurement with commands 0xE5, 0xF5, 0xE3, and 0xF3. The checksum byte is required for reading the electronic ID with commands 0xFA 0x0F and 0xFC 0xC9. For all other commands, the checksum byte is not supported.

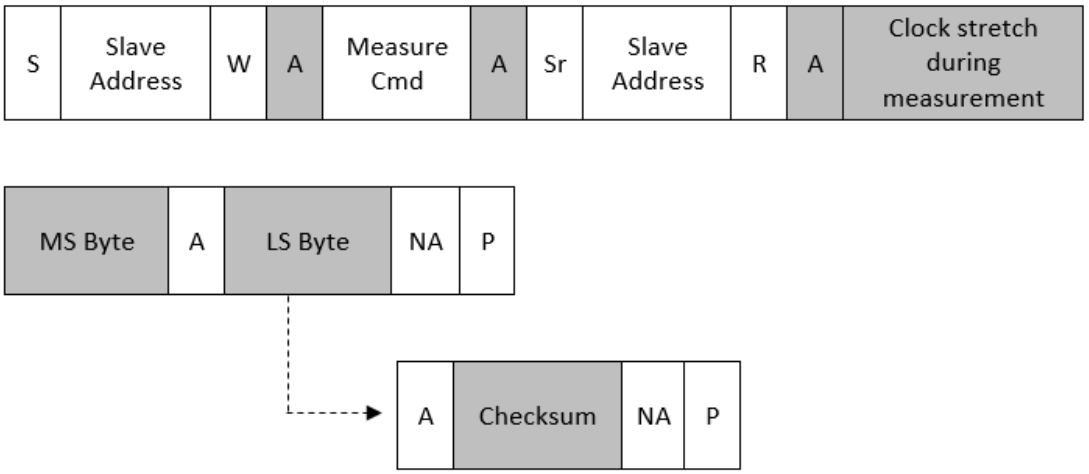
Table 12. I²C Bit Descriptions

Name	Symbol	Description
START	S	SDA goes low while SCL high.
STOP	P	SDA goes high while SCL high.
Repeated START	Sr	SDA goes low while SCL high. It is allowable to generate a STOP before the repeated start. SDA can transition to high before or after SCL goes high in preparation for generating the START.
READ	R	Read bit = 1
WRITE	W	Write bit = 0
All other bits	—	SDA value must remain high or low during the entire time SCL is high (this is the set up and hold time in Figure 1).

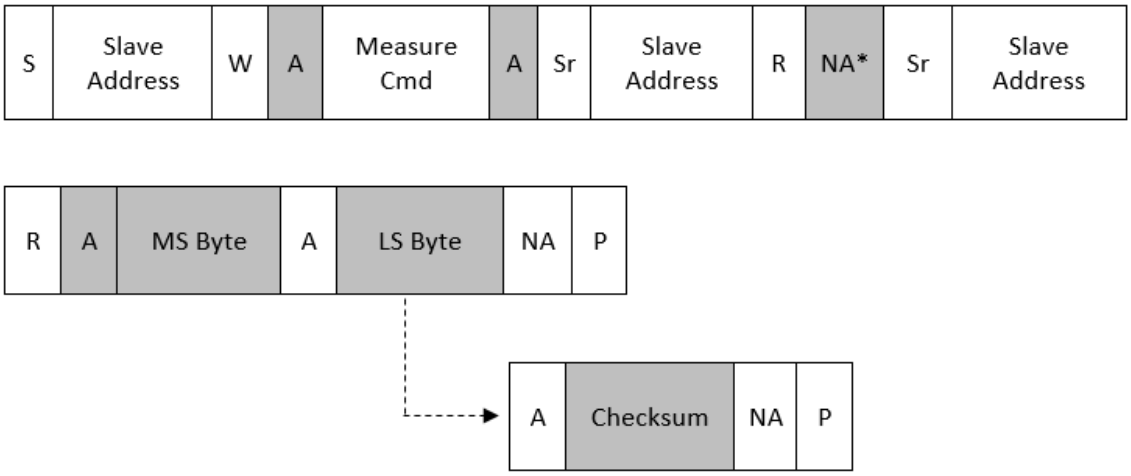
In the I²C sequence diagrams in the following sections, bits produced by the master and slave are color coded as shown:

Master	Slave

Sequence to perform a measurement and read back result (Hold Master Mode)



Sequence to perform a measurement and read back result (No Hold Master Mode)



***Note:** Device will NACK the slave address byte until conversion is complete.

5.1.1. Measuring Relative Humidity

Once a relative humidity measurement has been made, the results of the measurement may be converted to percent relative humidity by using the following expression:

$$\%RH = \frac{125 * RH\_Code}{65536} - 6$$

Where:

%RH is the measured relative humidity value in %RH

RH_Code is the 16-bit word returned by the Si7021

A humidity measurement will always return XXXXXX10 in the LSB field.

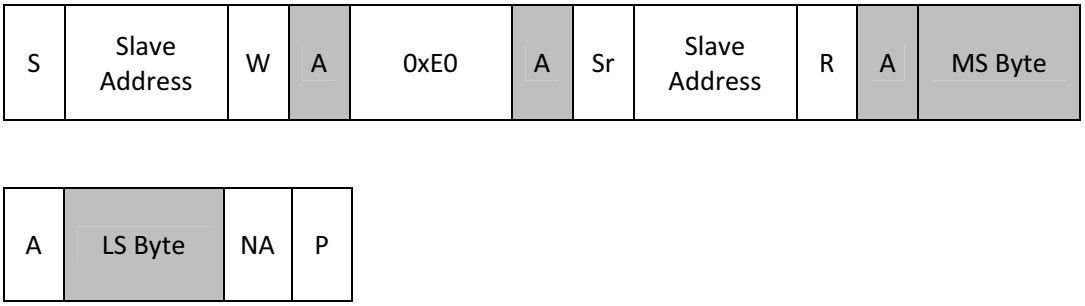
Due to normal variations in RH accuracy of the device as described in Table 4, it is possible for the measured value of %RH to be slightly less than 0 when the actual RH level is close to or equal to 0. Similarly, the measured value of %RH may be slightly greater than 100 when the actual RH level is close to or equal to 100. This is expected behavior, and it is acceptable to limit the range of RH results to 0 to 100%RH in the host software by truncating values that are slightly outside of this range.

5.1.2. Measuring Temperature

Each time a relative humidity measurement is made a temperature measurement is also made for the purposes of temperature compensation of the relative humidity measurement. If the temperature value is required, it can be read using command 0xE0; this avoids having to perform a second temperature measurement. The measure temperature commands 0xE3 and 0xF3 will perform a temperature measurement and return the measurement value, command 0xE0 does not perform a measurement but returns the temperature value measured during the relative humidity measurement.

The checksum output is not available with the 0xE0 command.

Sequence to read temperature value from previous RH measurement



The results of the temperature measurement may be converted to temperature in degrees Celsius (°C) using the following expression:

$$\text{Temperature (°C)} = \frac{175.72 * \text{Temp\_Code}}{65536} - 46.85$$

Where:

Temperature (°C) is the measured temperature value in °C

Temp_Code is the 16-bit word returned by the Si7021

A temperature measurement will always return XXXXXX00 in the LSB field.

5.2. Reading and Writing User Registers

There is one user register on the Si7021 that allows the user to set the configuration of the Si7021. The procedure for accessing that register is described below.

The checksum byte is not supported after reading a user register.

Sequence to read a register

S	Slave Address	W	A	Read Reg Cmd	A	Sr	Slave Address	R	A	Read Data	NA	P

Sequence to write a register

S	Slave Address	W	A	Write Reg Cmd	A	Write Data	A	P

5.3. Electronic Serial Number

The Si7021 provides a serial number individualized for each device that can be read via the I^2C serial interface.

Two I^2C commands are required to access the device memory and retrieve the complete serial number. The command sequence, and format of the serial number response is described in the figure below:

Master	Slave

First access:

S	Slave Address	W	ACK	0xFA	ACK	0X0F	ACK		
S	Slave Address	R	ACK						
	SNA_3	ACK	CRC	ACK	SNA_2	ACK	CRC	ACK	
	SNA_1	ACK	CRC	ACK	SNA_0	ACK	CRC	NACK	P

2nd access:

S	Slave Address	W	ACK	0xFC	ACK	0XC9	ACK
S	Slave Address	R	ACK				
	SNB_3	ACK	SNB_2	ACK	CRC	ACK	
	SNB_1	ACK	SNB_0	ACK	CRC	NACK	P

The format of the complete serial number is 64-bits in length, divided into 8 data bytes. The complete serial number sequence is shown below:

SNA_3	SNA_2	SNA_1	SNA_0	SNB_3	SNB_2	SNB_1	SNB_0

The SNB3 field contains the device identification to distinguish between the different Silicon Labs relative humidity and temperature devices. The value of this field maps to the following devices according to this table:

0x00 or 0xFF engineering samples

0x0D=13=Si7013

0x14=20=Si7020

0x15=21=Si7021

5.4. Firmware Revision

The internal firmware revision can be read with the following I^2C transaction:

S	Slave Address	W	A	0x84	A	0xB8	A	S	Slave Address

R	A	FWREV	NA	P

The values in this field are encoded as follows:

0xFF = Firmware version 1.0

0x20 = Firmware version 2.0

5.5. Heater

The Si7021 contains an integrated resistive heating element that may be used to raise the temperature of the sensor. This element can be used to test the sensor, to drive off condensation, or to implement dew-point measurement when the Si7021 is used in conjunction with a separate temperature sensor such as another Si7021 (the heater will raise the temperature of the internal temperature sensor).

The heater can be activated using HTRE, bit 2 in User Register 1. Turning on the heater will reduce the tendency of the humidity sensor to accumulate an offset due to "memory" of sustained high humidity conditions. Several different power levels are available. The various settings are adjusted using the Heater Control Register and are described in the following table.

Table 13. Heater Control Settings

HEATER[3:0]	Typical Current Draw* (mA)
0000	3.09
0001	9.18
0010	15.24
...	...
0100	27.39
...	...
1000	51.69
...	...
1111	94.20
***Note:** Assumes V_{DD} = 3.3 V.	

6. Control Registers

Table 14. Register Summary

Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
User Register 1	RES1	VDDS	RSVD	RSVD	RSVD	HTRE	RSVD	RES0
Heater Control Register	RSVD				HEATER[3:0]			

Notes:
1. Any register not listed here is reserved and must not be written. The result of a read operation on these bits is undefined.
2. Except where noted, reserved register bits will always read back as "1," and are not affected by write operations. For future compatibiliy, it is recommended that prior to a write operation, registers should be read. Then the values read from the RSVD bits should be written back unchanged during the write operation.

6.1. Register Descriptions

Register 1. User Register 1

Bit	D7	D6	D5	D4	D3	D2	D1	D0
Name	RES1	VDDS	RSVD	RSVD	RSVD	HTRE	RSVD	RES0
Type	R/W	R	R/W	R/W		R/W	R/W	R/W

Reset Settings = 0011_1010

Bit	Name	Function
D7; D0	RES[1:0]	Measurement Resolution: RH / Temp 00: 12 bit / 14 bit 01: 8 bit / 12 bit 10: 10 bit / 13 bit 11: 11 bit / 11 bit
D6	VDDS	V_{DD} Status: 0: V_{DD} OK 1: V_{DD} Low The minimum recommended operating voltage is 1.9 V. A transition of the V_{DD} status bit from 0 to 1 indicates that V_{DD} is between 1.8 V and 1.9 V. If the V_{DD} drops below 1.8 V, the device will no longer operate correctly.
D5, D4, D3	RSVD	Reserved
D2	HTRE	1 = On-chip Heater Enable 0 = On-chip Heater Disable
D1	RSVD	Reserved

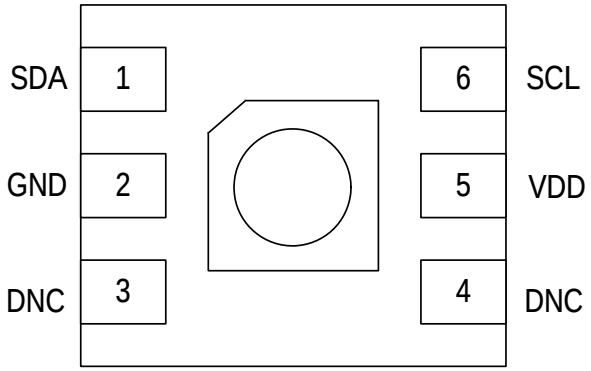
Register 2. Heater Control Register

Bit	D7	D6	D5	D4	D3	D2	D1	D0
Name	RSVD				Heater [3:0]			
Type	R/W				R/W			

Reset Settings = 0000_0000

Bit	Name	Function				
D3:D0	HEATER[3:0]	D3	D2	D1	D0	Heater Current
		0	0	0	0	3.09 mA
		0	0	0	1	9.18 mA
		0	0	1	0	15.24 mA
				...		
		0	1	0	0	27.39 mA
				...		
		1	0	0	0	51.69 mA
				...		
		1	1	1	1	94.20 mA
D7,D6, D5,D4	RSVD	Reserved				

7. Pin Descriptions: Si7021 (Top View)



Pin Name	Pin #	Pin Description
SDA	1	I^2C data
GND	2	Ground. This pin is connected to ground on the circuit board through a trace. Do not connect directly to GND plane.
VDD	5	Power. This pin is connected to power on the circuit board.
SCL	6	I^2C clock
DNC	3,4	These pins should be soldered to pads on the PCB for mechanical stability; they can be electrically floating or tied to V_{DD} (do not tie to GND).
T_{GND}	Paddle	This pad is connected to GND internally. This pad is the main thermal input to the on-chip temperature sensor. The paddle should be soldered to a floating pad.

8. Ordering Guide

Table 15. Device Ordering Guide

P/N	Description	Max. Accuracy		Pkg	Operating Range (°C)	Protective Cover	Packing Format
		Temp	RH				
Si7021-A20-GM	Digital temperature/ humidity sensor	±0.4 °C	± 3%	DFN 6	–40 to +85 °C	N	Cut Tape
Si7021-A20-GMR	Digital temperature/ humidity sensor	±0.4 °C	± 3%	DFN 6	–40 to +85 °C	N	Tape & Reel
Si7021-A20-GM1	Digital temperature/ humidity sensor	±0.4 °C	± 3%	DFN 6	–40 to +85 °C	Y	Cut Tape
Si7021-A20-GM1R	Digital temperature/ humidity sensor	±0.4 °C	± 3%	DFN 6	–40 to +85 °C	Y	Tape & Reel
Si7021-A20-IM	Digital temperature/ humidity sensor – industrial temp range	±0.4 °C	± 3%	DFN 6	–40 to +125 °C	N	Cut Tape
Si7021-A20-IMR	Digital temperature/ humidity sensor – industrial temp range	±0.4 °C	± 3%	DFN 6	–40 to +125 °C	N	Tape & Reel
Si7021-A20-IM1	Digital temperature/ humidity sensor – industrial temp range	±0.4 °C	± 3%	DFN 6	–40 to +125 °C	Y	Cut Tape
Si7021-A20-IM1R	Digital temperature/ humidity sensor – industrial temp range	±0.4 °C	± 3%	DFN 6	–40 to +125 °C	Y	Tape & Reel
Si7021-A20-YM0	Digital temperature/ humidity sensor – automotive	±0.4 °C	± 3%	DFN 6	–40 to +125 °C	N	Cut Tape
Si7021-A20-YM0R	Digital temperature/ humidity sensor – automotive	±0.4 °C	± 3%	DFN 6	–40 to +125 °C	N	Tape & Reel
Si7021-A20-YM1	Digital temperature/ humidity sensor – automotive	±0.4 °C	± 3%	DFN 6	–40 to +125 °C	Y	Cut Tape
Si7021-A20-YM1R	Digital temperature/ humidity sensor – automotive	±0.4 °C	± 3%	DFN 6	–40 to +125 °C	Y	Tape & Reel
Note: The “A” denotes product revision A and “20” denotes firmware version 2.0.							

9. Package Outline

9.1. Package Outline: 3x3 6-pin DFN

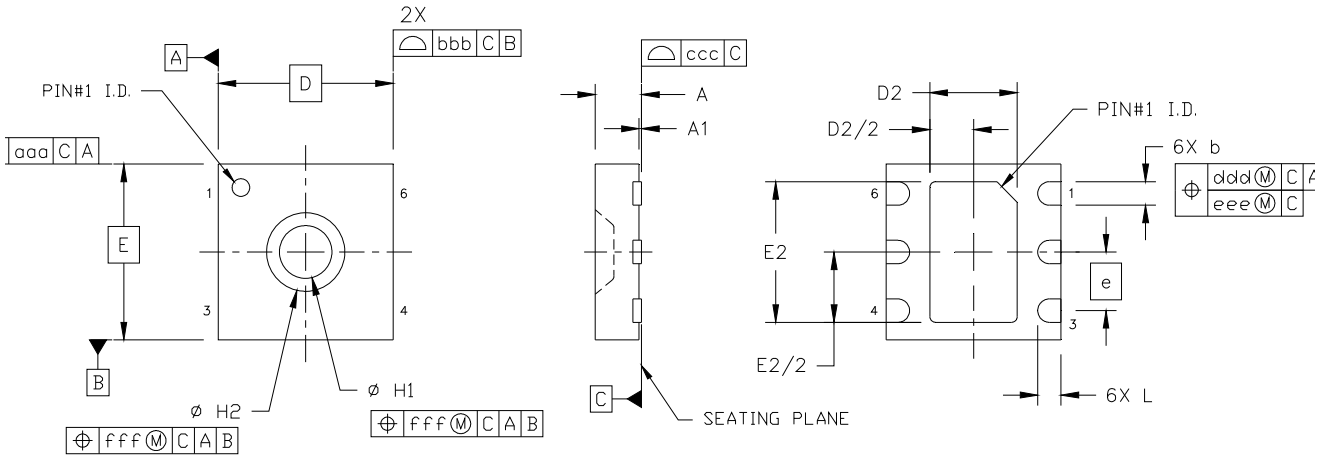


Figure 10. 3x3 6-pin DFN

Table 16. 3x3 6-pin DFN Package Diagram Dimensions

Dimension	Min	Nom	Max
A	0.70	0.75	0.80
A1	0.00	0.02	0.05
b	0.35	0.40	0.45
D		3.00 BSC.	
D2	1.40	1.50	1.60
e		1.00 BSC.	
E		3.00 BSC.	
E2	2.30	2.40	2.50
H1	0.85	0.90	0.95
H2	1.39	1.44	1.49
L	0.35	0.40	0.45
aaa		0.10	
bbb		0.10	
ccc		0.05	
ddd		0.10	
eee		0.05	
fff		0.05	

Notes:
1. All dimensions shown are in millimeters (mm).
2. Dimensioning and Tolerancing per ANSI Y14.5M-1994.

9.2. Package Outline: 3x3 6-pin DFN with Protective Cover

Figure 8 illustrates the package details for the Si7021 with the optional protective cover. The table below lists the values for the dimensions shown in the illustration.

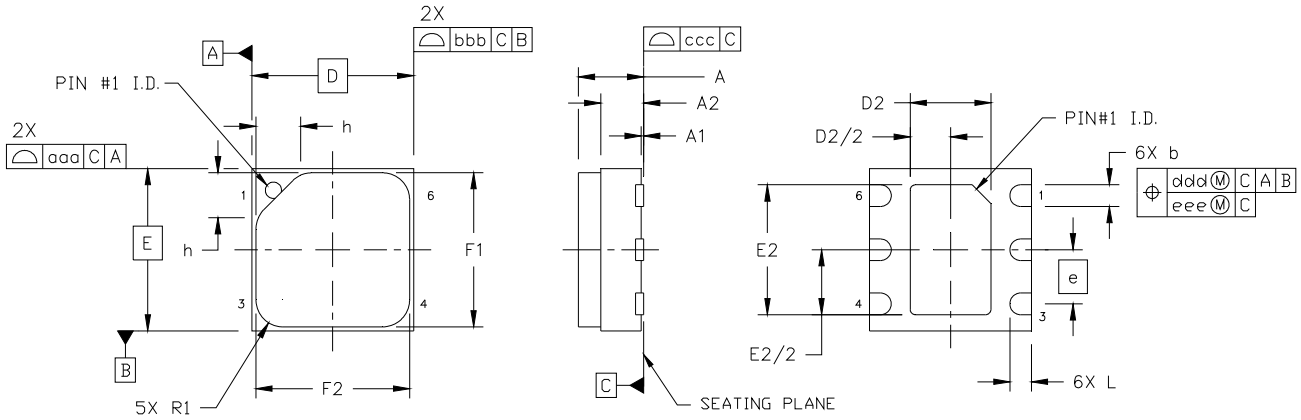


Figure 8. 3x3 6-pin DFN with Protective Cover

Table 17. 3x3 6-pin DFN with Protective Cover Package Diagram Dimensions

Dimension	Min	Nom	Max
A	—	—	1.21
A1	0.00	0.02	0.05
A2	0.70	0.75	0.80
b	0.35	0.40	0.45
D		3.00 BSC.	
D2	1.40	1.50	1.60
e		1.00 BSC.	
E		3.00 BSC.	
E2	2.30	2.40	2.50
F1	2.70	2.80	2.90
F2	2.70	2.80	2.90
h	0.76	0.83	0.90
L	0.35	0.40	0.45
R1	0.45	0.50	0.55
aaa		0.10	
bbb		0.10	
ccc		0.05	
ddd		0.10	
eee		0.05	

Notes:
1. All dimensions are shown in millimeters (mm).
2. Dimensioning and Tolerancing per ANSI Y14.5M-1994.

10. PCB Land Pattern and Solder Mask Design

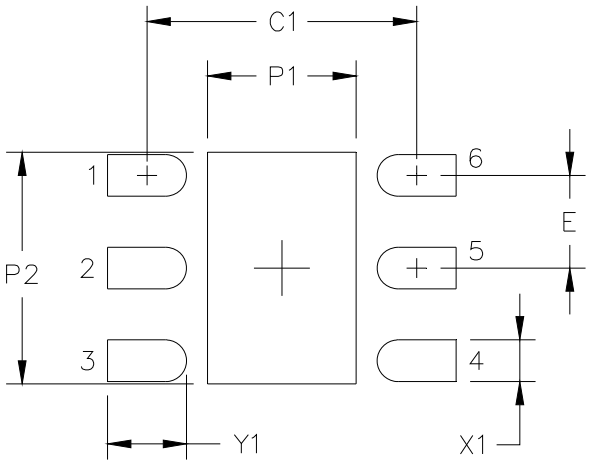


Figure 9. Si7021 PCB Land Pattern

Table 18. PCB Land Pattern Dimensions

Symbol	mm
C1	2.90
E	1.00
P1	1.60
P2	2.50
X1	0.45
Y1	0.85

Notes:

General

1. All dimensions shown are at Maximum Material Condition (MMC). Least Material Condition (LMC) is calculated based on a Fabrication Allowance of 0.05 mm.
2. This Land Pattern Design is based on the IPC-7351 guidelines.

Solder Mask Design

3. All metal pads are to be non-solder mask defined (NSMD). Clearance between the solder mask and the metal pad is to be 60 µm minimum, all the way around the pad.

Stencil Design

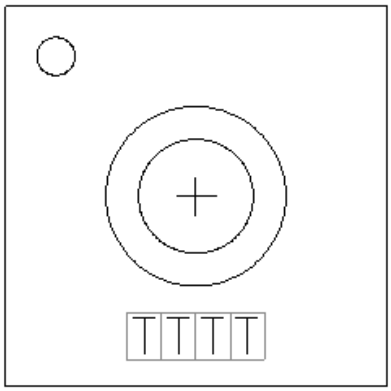
4. A stainless steel, laser-cut and electro-polished stencil with trapezoidal walls should be used to assure good solder paste release.
5. The stencil thickness should be 0.125 mm (5 mils).
6. The ratio of stencil aperture to land pad size should be 1:1 for all perimeter pins.
7. A 2x1 array of 1.00 mm square openings on 1.30 mm pitch should be used for the center ground pad to achieve a target solder coverage of 50%.

Card Assembly

8. The recommended card reflow profile is per the JEDEC/IPC J-STD-020 specification for Small Body Components.

11. Top Marking

11.1. Si7021 Top Marking



11.2. Top Marking Explanation

Mark Method:	Laser
Font Size	0.30 mm
Pin 1 Indicator:	Circle = 0.30 mm Diameter Upper-Left Corner
Line 1 Marking:	TTTT = Mfg Code

12. Additional Reference Resources

- AN607: Si70xx Humidity Sensor Designer's Guide

Document Change List

Revision 0.9 to Revision 0.91

- Updated Table 2 on page 4.

Revision 0.91 to Revision 1.0

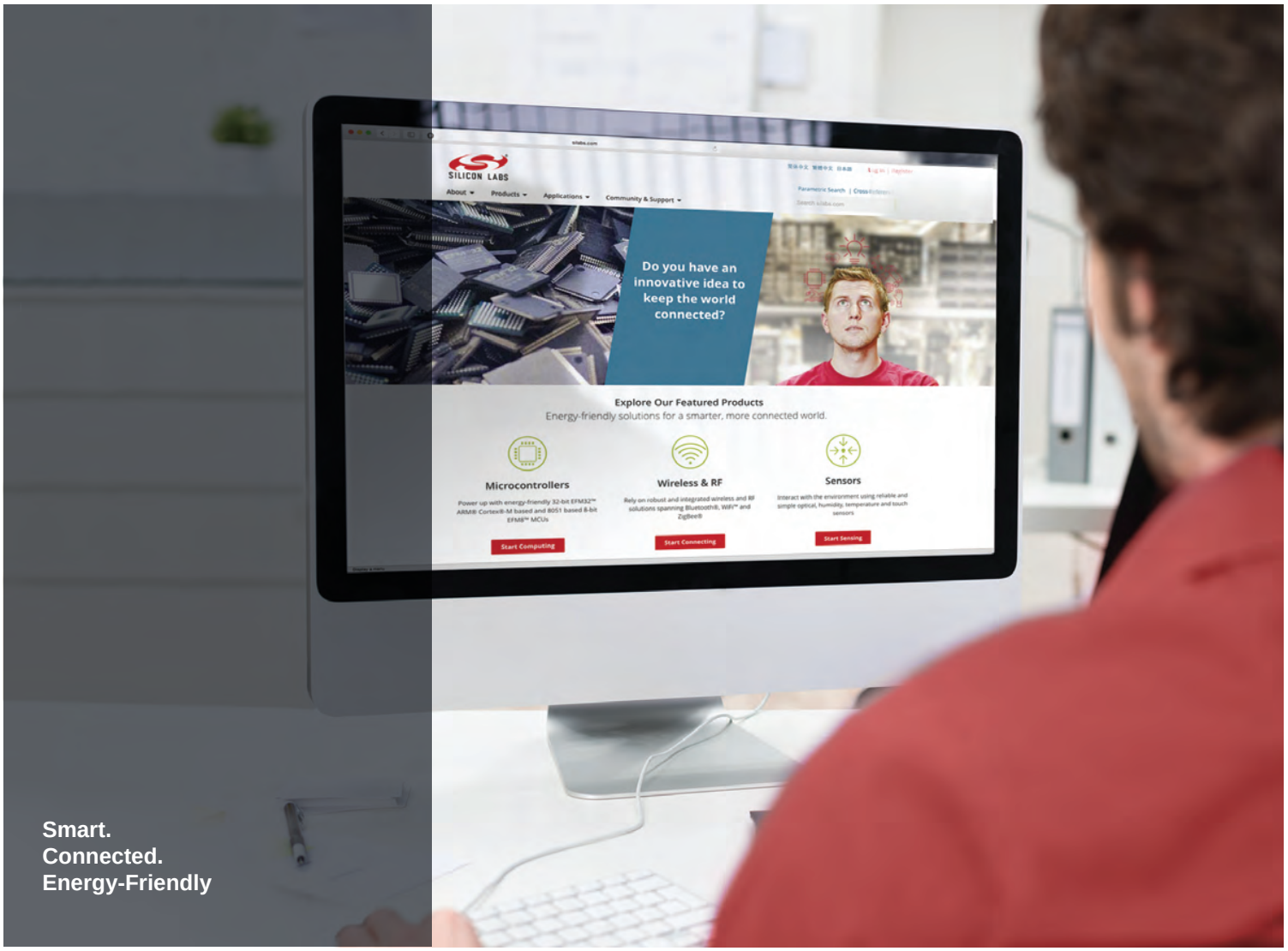
- Updated document revision to 1.0.

Revision 1.0 to Revision 1.1

- Updated Footnote 2 in Table 3.
- Updated Section “4.5. Protecting the Sensor” .
- Updated Table 9.
- Corrected a typo in the I^2C sequence for no-hold mode in Section “5. I2C Interface” .
- Corrected a typo in Table 12.
- Updated Table 17 dimensions F1 and F2.

Revision 1.1 to Revision 1.2

- Updated diagram in "5.4. Firmware Revision" on page 24.
- Updated notes in Table 18, “PCB Land Pattern Dimensions,” on page 32.
- Changed packing format from tube to cut tape for all non-tape & reel part numbers without protective filter covers.



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com