ECSE 1010

Class #25: Digital Electronics and Software
Python2.7-Based Control and Data Acquisition

Purpose: In this experiment we will learn to use the Python 2.7 Programming Language to provide input signals for Analog Discovery and also to monitor and record output signals.

Background: Before doing this experiment, students should be able to

- Review online background materials.
- Build and operate simple circuits on a Protoboard.
- Measure the voltages and determine the currents using a math channel in simple Protoboard circuits using Analog Discovery
- Analyze simple circuits consisting of combinations of resistors, especially voltage dividers.
- Do a transient (time dependent) simulation of circuits using LTspice
- Plot data and perform other basic functions using Python.
- Review the background for the previous experiments.

Learning Outcomes: Students will be able to

- Install Python support files for Analog Discovery.
- Use Python to communicate with Analog Discovery.
- Use Python to generate output signals for Analog Discovery.
- Use Python to collect measured data from Analog Discovery.
- Modify Python code to implement a simple control process.

Resources Required:

- LTspice
- Analog Discovery and Parts Kit
- A partner – It is best to do this experiment as a team.
- Python 2.7 and some pre-written code (see next page).

Helpful links for this experiment can be found on the course website under Class #25.

Pre-Lab
Required Reading: Before beginning the lab, at least one team member must read over and be generally acquainted with this document and the other **required reading** materials.
Hand-Drawn Circuit Diagrams: Before beginning the lab, hand-drawn circuit diagrams must be prepared for all circuits either to be analyzed using LTspice or physically built and characterized using Analog Discovery.
Due: At the beginning of Class #27

Background Preparation:
Download **Python 2.7**
https://www.python.org/downloads/ (Use Python 2.7 version not 3.0)

(The following is included for completeness, because you should already have done this.)
Download the Waveform software from
http://store.digilentinc.com/waveforms-2015-download-only/

In the folder ***Digilent/WaveFormsSDK*** that installs on your computer with WaveForms, there is a **WaveForms SDK Reference Manual.pdf** that lists the different commands to Analog Discovery
.
In the folder ***Digilent/WaveFormsSDK/samples/py*** are sample programs to use.
You will need the file **dwfconstants.pyc** (or dwfconstants.py) so that file needs to be copied to the folder that you will be creating your Python programs in.

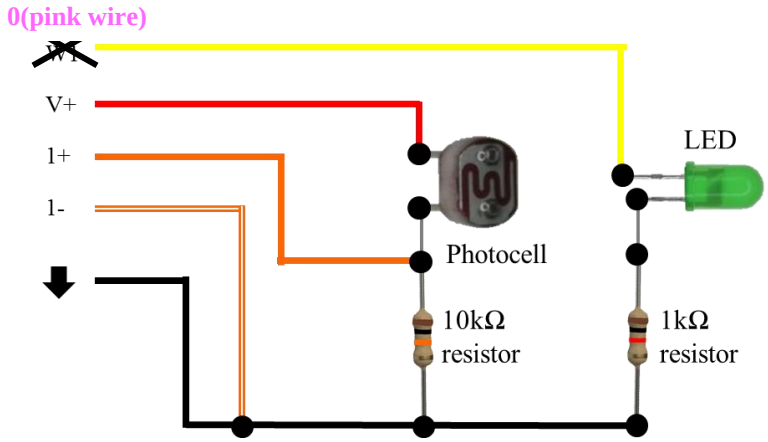
ECSE 1010

In this experiment (originally written by Jeff Rodriguez of Anderson High School in Cincinnati, but modified for our purposes), we will be using Analog Discovery and Python2.7 to design a simple automatic nightlight (turns on and off automatically and adjusts the brightness based on the surrounding light level). Jeff used Matlab to control analog input and output functions on Analog Discovery (version 1). That option is not available with Analog Discovery 2.

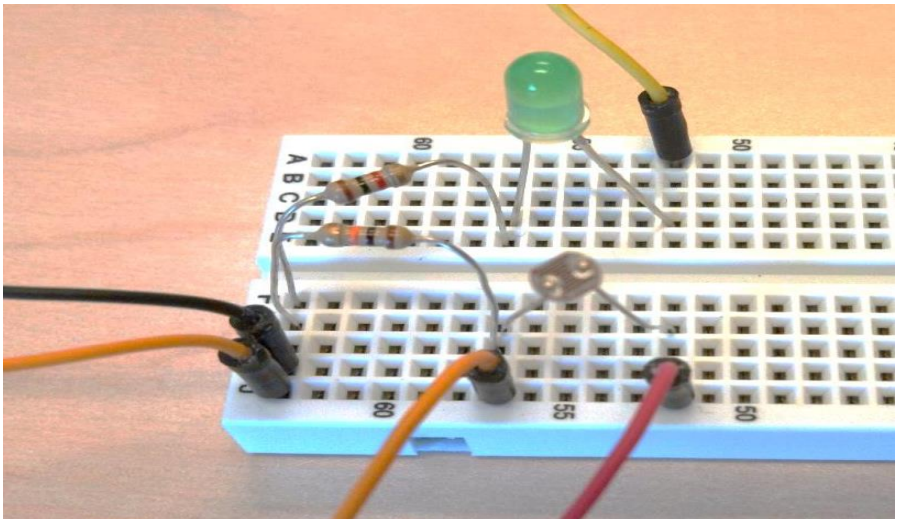
This idea is similar to what your phone or computer may do when it adjusts the brightness of the screen based on whether you are out in the sun or in a dimly lit room. When you are out in the sun, the ambient light level is usually high, which causes the device to increase the brightness of the screen so you can see it better. When you are in a dimly lit room, it can be very difficult to read a screen if it is at full brightness, so the device may set the screen to a lower brightness level. We will be doing the opposite: making an LED brighter as the room gets darker.

Part A: Building the Circuit

Start with the circuit shown at www.Rodshome.com lab 8, BUT change the yellow wire to attach to digital I/O port 0. It is a pink wire from the Analog Discovery. (Matlab can only control analog functions on Analog Discovery 1, so Jeff Rodriquez had to use one of the function generator outputs for his PWM signal.) All partners should verify that the circuit is built correctly.

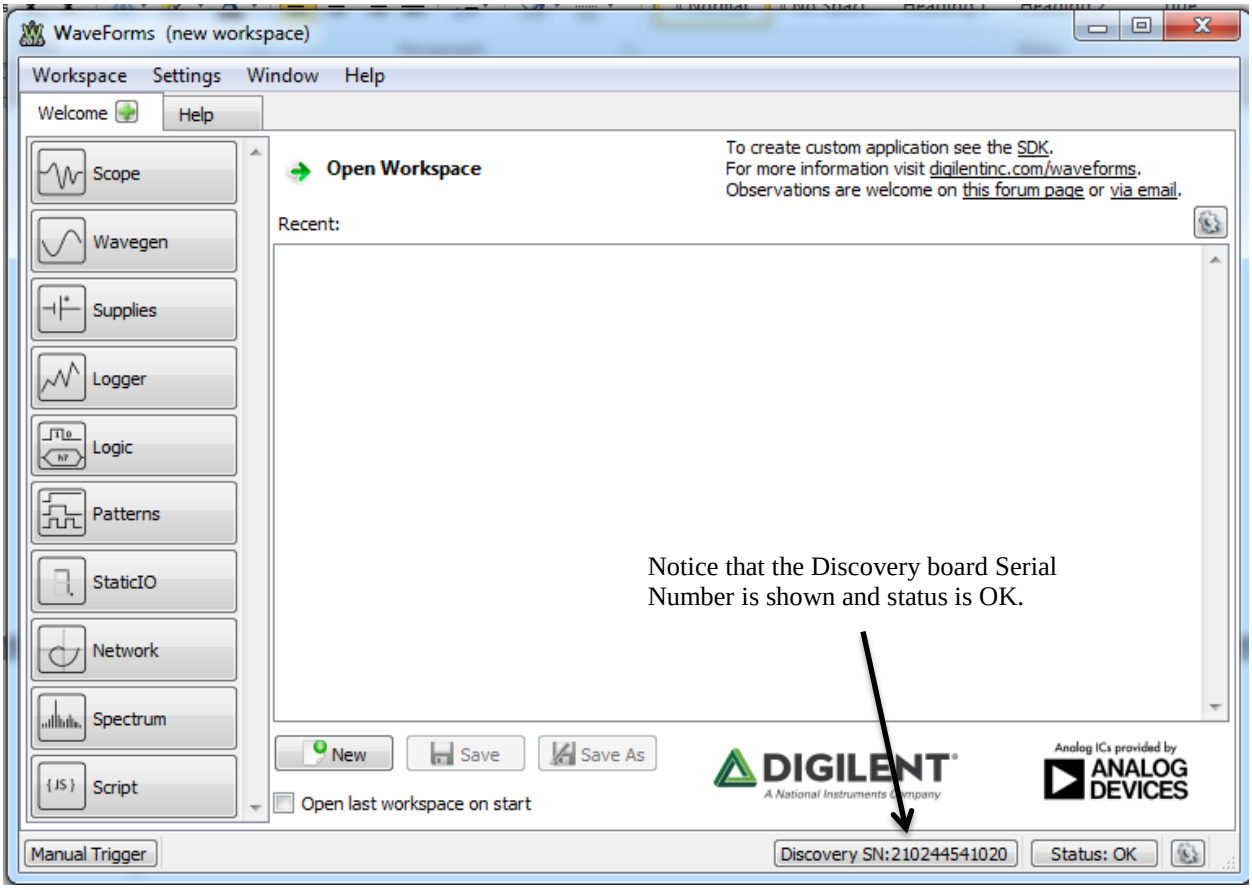


Pay close attention to the color code on the resistors. The 1 kΩ (Brown-Black-Red) should be connected in series with the LED. The 10 kΩ (Brown-Black-Orange) should be connected in series with the photocell.



Part B: Software Check for Analog Discovery

Run the Waveforms software. If you installed everything correctly, you should see the image below indicating the Analog Discovery board is recognized.



Turn on both the Scope and the +5V supply. On the Scope, turn off any triggering. On the Scope you should see a nearly DC signal that changes when the light detected by the photocell changes. This verifies that the detector part of the simple circuit is working. Then, turn on StaticIO. Make Digital channel 0 a switch and set it to high. The LED should turn on and off with the switch.

Measure the voltage across the 10kΩ resistor with the photocell covered and uncovered. You have done something similar in a previous experiment, so it is a good idea to check your results.

Maximum Light	[V]
Minimum Light	[V]

Measure the voltage across the 1kΩ resistor when the LED is on.

$V_{1k\Omega}$	[V]

Once you have confirmed both parts of the circuit are working, exit Waveforms software. We will only be using Waveforms again, briefly, in Part D of this experiment.

Part C: Simple Input/Output Commands

- Wire the circuit as shown in Part A.
- Open IDLE (Python GUI that downloads with Python 2.7. It is reasonably simple to use but you can search for resources on using IDLE if you have any questions.)
- Click on File > New and type the following (or download from the course webpage) save as "softwarecheck1.py".

Note: Be careful with indentations.

```
"""
  Requires:
     Python 2.7
"""
from ctypes import *
from dwfconstants import *
import time
import sys
import math


SAMPLES = 10 #This is the number of samples of the analog channel
SLEEP_TIME=1 #This is the time between samples

#--------------------------------------------------
# using windows plateform
dwf = cdll.dwf

#declare ctype variables
hdwf = c_int()
voltage = c_double()
hzSys = c_double()

#print DWF version
version = create_string_buffer(16)
dwf.FDwfGetVersion(version)
print "DWF Version: "+version.value

#open device
print "Opening first device..."
dwf.FDwfDeviceOpen(c_int(-1), byref(hdwf))

if hdwf.value == hdwfNone.value:
   print "failed to open device"
   quit()

# enable positive supply
dwf.FDwfAnalogIOChannelNodeSet(hdwf, c_int(0), c_int(0), c_double(True))
# set voltage to 5 V
dwf.FDwfAnalogIOChannelNodeSet(hdwf, c_int(0), c_int(1), c_double(5.0))
# master enable
dwf.FDwfAnalogIOEnableSet(hdwf, c_int(True))

# enable output/mask on 8 LSB IO pins, from DIO 0 to 7
dwf.FDwfDigitalIOOutputEnableSet(hdwf, c_int(0x00FF))
```

```
#Set bit0 this will turn on the LED
dwf.FDwfDigitalIOOutputSet(hdwf, c_int(0x01))
#leave it on for 5 seconds
time.sleep(5)
#reset bit0 this will turn off LED
dwf.FDwfDigitalIOOutputSet(hdwf, c_int(0x00))
"""
print "Preparing to read sample..."

dwf.FDwfAnalogInChannelEnableSet(hdwf, c_int(0), c_bool(True))
dwf.FDwfAnalogInChannelOffsetSet(hdwf, c_int(0), c_double(0))
#I set this to 10 not 5.
dwf.FDwfAnalogInChannelRangeSet(hdwf, c_int(-1), c_double(10))
dwf.FDwfAnalogInConfigure(hdwf, c_bool(False), c_bool(False))
for i in range(SAMPLES):
  time.sleep(SLEEP_TIME)
  dwf.FDwfAnalogInStatus(hdwf, False, None)
  dwf.FDwfAnalogInStatusSample(hdwf, c_int(0), byref(voltage))
  print "Voltage:  " + str(voltage.value)

"""

#---End the program--------
dwf.FDwfDigitalOutReset(hdwf);
dwf.FDwfDeviceCloseAll()
```

Note: Multi-line comments are enclosed in triple double quotes, “”””.

- Click on Run > Run module or click on F5

Record what happens to the LED.

- In the Python program change the following line of code
  dwf.FDwfDigitalIOOutputSet(hdwf, c_int(0x01))
  from c_int(0x01) to c_int(0x02)
- Execute the program

Record what happens to the LED.

- Rewire the high side (long leg) of the LED to digital output 1 on the Analog Discovery (green wire).
- Execute program.

Record what happens to the LED.

- Return the wiring to Digital I/O port 0 as shown in Part A.
- In the Python program change the following line of code back to the original
  dwf.FDwfDigitalIOOutputSet(hdwf, c_int(0x01))

What value would need to be written in the above line of code to turn on LEDs if they were wired to digital ports 0, 1 and 2?

- Wire two additional LEDs and resistors on your proto-board and connect them to ports 1 (green wire) and 2 (purple wire.) You should have three resistor-LED combinations, each connected to one of the three ports at one end and grounded on the other. Hand draw your modified circuit diagram.
- Change the same line of code to the hex value you stated in the above to control, in turn, each of the three digital ports you are using. (Replace the question marks with the correct hex value.)
  dwf.FDwfDigitalIOOutputSet(hdwf, c_int(0x??))

What did you observe? Demonstrate control of digital port 2 to a TA or instructor _____

- Remove the two sets of triple double quotes.
- Save and Run the program
- After the LED is on for 5 seconds the voltage outputted by the photo resistor will be displayed.
- As it is printing out cover the photo resistor and see the changes in the voltage.

Record the voltages printed to the screen. If any are zero or negative, check your circuit.

Part D: Introduction to PWM in Python

To control the brightness of an LED you can vary the power to the LED. The more power the brighter the LED. The less power the dimmer the LED. We can 'simulate' varying levels of power by oscillating the output from the Analog Discovery to the LED. If we turn the LED on 50% of the time and turn it off 50% of the time, it appears half as bright as if the LED is on 100% of the time.

The programmer needs to calculate the DIVIDER for the desired frequency of the PWM on port 0 for the program listed below.

Internal Clock Frequency=DIVIDER * Desired Frequency *(count_pulse_low +count_pulse_high)

(count_pulse_low +count_pulse_high), which is called lowV and highV in the program and will sum to 100.

$$DIVIDER = \frac{Internal\ Clock\ Frequency}{100 * Desired\ Frequency}$$

All frequencies are in Hz.

- Return the circuit to the same wiring you had for part A, with only one resistor-LED combo.
- Open IDLE (Python GUI)
- Click on File > New and type the following (or download from the course webpage) save as "AnalogDiscoveryPWM.py".
- Click on Run > Run module or click on F5

```
"""
  Requires:
     Python 2.7
"""
from ctypes import *
from dwfconstants import *
import time
import sys
import math
#---Variables for user to set----------------------
SLEEP_TIME=1
SAMPLES=10
BRIGHT=4.5
DARK =2.0
DIVIDER=500
#-----------------------------------------------
# using windows plateform
dwf = cdll.dwf

#declare ctype variables
hdwf = c_int()
voltage = c_double()
hzSys = c_double()

#print DWF version
version = create_string_buffer(16)
dwf.FDwfGetVersion(version)
print "DWF Version: "+version.value

#open device
print "Opening first device..."
dwf.FDwfDeviceOpen(c_int(-1), byref(hdwf))

if hdwf.value == hdwfNone.value:
   print "failed to open device"
   quit()

# enable positive supply
dwf.FDwfAnalogIOChannelNodeSet(hdwf, c_int(0), c_int(0), c_double(True))
# set voltage to 5 V
dwf.FDwfAnalogIOChannelNodeSet(hdwf, c_int(0), c_int(1), c_double(5.0))
"""# master enable
dwf.FDwfAnalogIOEnableSet(hdwf, c_int(True))
#-----------------------------------------------
#--Set up analog input on channel 1---------------
dwf.FDwfAnalogInChannelEnableSet(hdwf, c_int(0), c_bool(True))
dwf.FDwfAnalogInChannelOffsetSet(hdwf, c_int(0), c_double(0))
dwf.FDwfAnalogInChannelRangeSet(hdwf, c_int(-1), c_double(10))
dwf.FDwfAnalogInConfigure(hdwf, c_bool(False), c_bool(False))
#-----------------------------------------------"""
#The function FDwfDigitalOutInternalClockInfo is used to retrieve the internal clock frequency.
#and store frequency in variable hzSys
dwf.FDwfDigitalOutInternalClockInfo(hdwf, byref(hzSys))
print "the internal clock frequency is "+str(hzSys.value)
#----- pulse on IO pin 0 ----------------------
#The function FDwfDigitalOutEnableSet enables or disables the channel(second argument)
```

```
dwf.FDwfDigitalOutEnableSet(hdwf, c_int(0), c_int(1))

#Set the value of the divider on digital I/O port 0
dwf.FDwfDigitalOutDividerSet(hdwf, c_int(0), c_int(int(DIVIDER)))
#-------------------------------------------------
print " Duty cycle of 100%"
dwf.FDwfDigitalOutCounterSet(hdwf, c_int(0), c_int(1), c_int(100))
#The function FDwfDigitalOutConfigure is used to start or stop the instrument.
dwf.FDwfDigitalOutConfigure(hdwf, c_int(1))
time.sleep(5)
#-------------------------------------------------
print "Duty cycle of 50%"
dwf.FDwfDigitalOutCounterSet(hdwf, c_int(0), c_int(50), c_int(100))
#The function FDwfDigitalOutConfigure is used to start or stop the instrument.
dwf.FDwfDigitalOutConfigure(hdwf, c_int(1))
time.sleep(5)
# turn off light
dwf.FDwfDigitalOutCounterSet(hdwf, c_int(0), c_int(100), c_int(1))
dwf.FDwfDigitalOutConfigure(hdwf, c_int(1))

ticks=input ("How many clock ticks would you like the light on (1 - "+str(hzSys.value)+ ")")
onlite=int(float(ticks/hzSys.value)*100.0)
if (onlite>100):
   onlite=100
   offlite=1
else:
   offlite=100-onlite
if (offlite<1):
   offlite=1
if (onlite<1):
   onlite=1
print "The duty cycle is "+str((ticks/hzSys.value)*100) +"%"
dwf.FDwfDigitalOutCounterSet(hdwf, c_int(0), c_int(offlite), c_int(onlite))
#The function FDwfDigitalOutConfigure is used to start or stop the instrument.
dwf.FDwfDigitalOutConfigure(hdwf, c_int(1))
time.sleep(5)
#---End the program--------
dwf.FDwfDigitalOutReset(hdwf);
dwf.FDwfDeviceCloseAll()
```

Record the internal clock frequency

Describe what happens to the LED.

When the program is running, you will be asked a question. What is the question and what is the answer to produce a duty cycle of 75%?

Part D: Nightlight Setup

The nightlight will function by progressively turning on the LED as the measured light decreases. In your circuit, as the ambient light level decreases, the voltage you measure for the photocell portion of the circuit will also decrease. For this system, we want to have 11 light levels (1 off level and 10 different levels of brightness when turned on.) Before we can create the code to do this, we first need to test our system to determine the voltage range over which the photocell will vary and possibly adjust the value of the resistor in series with the photocell.

You will make measurements using both software tools to validate the use of Python

- Open Waveforms and then the Supplies and Logger windows. Turn on the 5V power.
- Use the Logger or a Voltmeter to measure the voltage across the 10kΩ resistor when the photocell is illuminated by the light in the room. Record the value below.

Voltage level of ambient light: _____

- One of your team will now need to place their fingers over the photocell, covering as much of the device as possible. Use the Logger or a Voltmeter to measure the voltage across the 10kΩ resistor when photocell is in the dark. Record the value below.

Voltage level of darkness: _____

This experiment was designed for a specific photocell so that the voltages measured are in the range from 2V to 5V and to use as much of the range as possible for good sensitivity. If you do not have optimum conditions, the 10kΩ resistor in series with the photocell may be too large or too small. Using your knowledge of voltage dividers, adjust the value of this resistor until the measured voltage meets both criteria. You should have maximum and minimum voltages within a few tenths of a volt of 5V and 2V, respectively. Record your new fixed resistor valueyou're your measured voltages below.

New fixed resistor value: _____

Voltage level of ambient light: _____

Voltage level of darkness: _____

Once you have realized this goal, turn off Waveforms.

We will now return control to Python.

- Use the same wiring as you did for part A, with a single resistor-LED combo.
- Open IDLE (Python GUI)
- Click on File > New and type the following (or download from the course webpage) save as "ExamplePWM.py".
- If necessary change the following values in the program:
  - ***SLEEP_TIME=1***
  - ***SAMPLES=10***
  - ***BRIGHT=4.5***
  - ***DARK =2.0***
  - ***DIVIDER=500***

- Click on Run > Run module or click on F5

```
"""
  Requires:
     Python 2.7
"""
from ctypes import *
from dwfconstants import *
import time
import sys
import math
#---Variables for user to set----------------------
SLEEP_TIME=1
SAMPLES=10
BRIGHT=4.5
DARK =2.0
DIVIDER=500
#-----------------------------------------------
# using windows plateform
dwf = cdll.dwf

#declare ctype variables
hdwf = c_int()
voltage = c_double()
hzSys = c_double()

#print DWF version
version = create_string_buffer(16)
dwf.FDwfGetVersion(version)
print "DWF Version: "+version.value

#open device
print "Opening first device..."
dwf.FDwfDeviceOpen(c_int(-1), byref(hdwf))

if hdwf.value == hdwfNone.value:
    print "failed to open device"
    quit()

# enable positive supply
dwf.FDwfAnalogIOChannelNodeSet(hdwf, c_int(0), c_int(0), c_double(True))
# set voltage to 5 V
dwf.FDwfAnalogIOChannelNodeSet(hdwf, c_int(0), c_int(1), c_double(5.0))
# master enable
dwf.FDwfAnalogIOEnableSet(hdwf, c_int(True))
#-----------------------------------------------
#--Set up analog input on channel 1---------------
dwf.FDwfAnalogInChannelEnableSet(hdwf, c_int(0), c_bool(True))
dwf.FDwfAnalogInChannelOffsetSet(hdwf, c_int(0), c_double(0))
dwf.FDwfAnalogInChannelRangeSet(hdwf, c_int(-1), c_double(10))
dwf.FDwfAnalogInConfigure(hdwf, c_bool(False), c_bool(False))
#-----------------------------------------------
#The function FDwfDigitalOutInternalClockInfo is used to retrieve the internal clock frequency.
#and store frequency in variable hzSys
dwf.FDwfDigitalOutInternalClockInfo(hdwf, byref(hzSys))
print "the internal clock frequency is "+str(hzSys.value)
#----- pulse on IO pin 0 ----------------------
#The function FDwfDigitalOutEnableSet enables or disables the channel(second argument)
```

```
dwf.FDwfDigitalOutEnableSet(hdwf, c_int(0), c_int(1))

#Set the value of the divider on digital I/O port 0
dwf.FDwfDigitalOutDividerSet(hdwf, c_int(0), c_int(int(DIVIDER)))
#-------------------------------------------------
print "preparing to read sample..."
for i in range(SAMPLES):
  time.sleep(SLEEP_TIME)
  dwf.FDwfAnalogInStatus(hdwf, False, None)
  dwf.FDwfAnalogInStatusSample(hdwf, c_int(0), byref(voltage))
  print "Voltage:  " + str(voltage.value)
  volts=voltage.value
  highV=int(((BRIGHT-volts)/(BRIGHT-DARK))*100)
  if(highV<=0):
     highV=1
  lowV=(100-highV)
  if(lowV<=0):
     lowV=1
#-------------------------------------------------
#duty pulse
  dwf.FDwfDigitalOutCounterSet(hdwf, c_int(0), c_int(lowV), c_int(highV))
#-------------------------------------------------

#The function FDwfDigitalOutConfigure is used to start or stop the instrument.
  dwf.FDwfDigitalOutConfigure(hdwf, c_int(1))
#---End the program--------
dwf.FDwfDigitalOutReset(hdwf);
dwf.FDwfDeviceCloseAll()
```

Record the voltage readings and what happens to the LED when you cover the photocell with your hands and then when it gets direct light. Demonstrate the operation of your nightlight to a TA or instructor. Also show them any modifications you have made to your circuit and Python code. _____

Part E: Nightlight Code

Describe the final version of your Python code. In particular, how does it use the measured voltage across the fixed photocell resistor to set the intensity of the light from the LED?

Paste a copy of your final code in your report.

Part F: Reflection

Take a moment to reflect on what you have learned in this experiment, then discuss what you have learned. In particular, discuss the Python codes you have used and what they do, the use of a photocell to measure light level and how the final version of your nightlight might be different in a very bright room. Discuss anything that still confuses you and/or anything else you would like to know about these topics and any other experiments or simulations you think might be worth doing. Discuss how well your final circuit and code worked and any problems you may have had. Include the reflections in your report.