NATURAL TIMERS FOR IOT

A Design Project Report

Presented to the School of Electrical and Computer Engineering of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering, Electrical and Computer Engineering

Submitted by

Chris Yang

MEng Field Advisor: Dr. Van Hunter Adams

Degree Date: May 2024

Abstract

Master of Engineering Program
School of Electrical and Computer Engineering
Cornell University
Design Project Report

Project Title: Natural Timers for IoT

Author: Chris Yang

Abstract: This project explores replacing timer peripherals with natural triggers in IoT applications. In many IoT applications, sensors are often paired with a real-time clock (RTC) peripheral for scheduling measurements on sleep/wake cycles. Traditional IoT devices would implement a sleep/wake cycle in which the microcontroller is asleep for a timed duration, has a scheduled wake-up, and takes a sensor measurement. Upon waking up, the microcontroller completes a desired task (e.g. logging information) and goes back to sleep. The problem with this is that in many microcontrollers like the RP2040-based Raspberry Pi Pico, the sleep mode is still very power-hungry and would still produce poor battery longevity. This project aims to replace this sleep/wake cycle with a dead/alive cycle in which the microcontroller is at a completely off or “dead” state with zero power consumption when waiting for a natural trigger. Using this notion, this project integrates a piezo sensor with a latching circuit that allows us to use a natural process to replace a timer in an IoT system.

Executive Summary

This project was completed in the span of two semesters. In the first semester, my partner Michael Awad and I first investigated and confirmed the power consumption while in sleep mode on the Raspberry Pi Pico. Using a Nordic Power Profiler Kit II, measured the current draw of the Pico in different modes of operation. The sleep-mode power consumption offered a baseline against which to compare our lower-power solution. Next, we proved the working concept of a microcontroller wake-up latching circuit and introduced dead/alive mode. We demonstrated the latching circuit working as an external current source would be able to wake up the Raspberry Pi Pico microcontroller from a powered-off state. However, this was completed with an ideal power supply generated voltage source.

This semester, I focused on developing a proof-of-concept circuit into a field-deployable system. Therefore, for the natural current source, we chose to use a piezoelectric film sensor, as this could detect vibrations made in nature such as a gust of wind or a bird pecking from a birdfeeder. This was quite tricky because a piezo sensor behaves substantially differently compared to an ideal voltage source. After multiple circuit modifications, the piezo sensor was able to latch the microcontroller on successfully. Next, we had to decide what we could do once the microcontroller woke up. Given that we wanted to characterize these natural processes as timers, we decided that it was most appropriate to incorporate the logging of time on a microSD card. This was done by having the Pico set a GPIO pin high once the microcontroller woke up and have another Raspberry Pi Pico (acting as a computer) permanently running with an RTC to poll on that pin to record what time of day the trigger woke up the first Pico. Hence, once the piezo sensor was triggered, it would wake up the first Pico, setting a GPIO pin high, which would latch the microcontroller off and store data to an SD card via the other Pico. Once all of that was working, an experiment was conducted by placing this setup in an outdoor environment (outside of Phillips Hall on Cornell campus) for a whole day. I would come back a day later to find a log file on the SD card to see exactly what time a natural trigger woke the microcontroller up. Upon opening the log file, we found that the microcontroller woke up a total of four times, two times in the afternoon when wind speeds were at their peak, and twice at night.

Introduction

As discussed earlier, the first task of the project was to investigate the sleep mode on the Raspberry Pi Pico. The Raspberry Pi Pico's datasheet included expected average current measurements in the sleep mode, shown in Figure 1 below.

Pico board	VBUS current @5V (mA)		
	Temperature (°C)		
	-25	25	85
#1	1.35	1.30	1.81
#2	1.53	1.39	1.92
#3	1.40	1.32	1.92
Mean	**1.4**	**1.3**	**1.9**

Figure 1:Average Current in Sleep Mode from Pico Datasheet

As seen in the table, we can expect an average of 1.3mA of current consumption in sleep mode. This is significantly higher than that of most microcontrollers, as sleep modes of other microcontrollers typically have current values of orders of microamps (µA) compared to a whole milliamp (mA) on the Pico. After running the hello_sleep program on the Pico, we indeed confirmed this to be accurate, measuring a similar average current value of 1.32mA, as shown below in Figure 2.

Figure 2: Current Consumption of Pico in Sleep Mode from Power Meter

After confirming our measurements against the datasheet, we explored the use of dead/alive mode and analyzed the potential power-saving advantages we could get.

We can model the average current consumption with the following equation:

$$I_{avg} = I_{avg\_off} + I_{avg\_on} = (\alpha_{off} * I_{off}) + (\alpha_{on} * I_{on})$$

The average current consumption is equivalent to the average current in the off state summed with the average current in the on state. The average current in either state is calculated by the proportion of time (out of 1) the microcontroller stays in that state. For example, if the microcontroller stays off for 90% of the time, α_{off} would equal 0.9 and α_{on} would equal 0.1. Once we figure out the average current draw, we divide the battery capacity (in mAh) by the current consumption to get battery life in number of hours.

For this comparison, we will do two calculations comparing the battery life of sleep mode vs. dead/alive mode for when the microcontroller wakes up every 10 seconds and every hour. Since the microcontroller latches itself off instantly after it wakes up, we can assume the time on is about 0.5 seconds. Let's also assume $I_{on} = 40mA$, as this was the current measured when the microcontroller wakes up. The most viable battery power source is a AA battery, which has a 2850mAh capacity.

For say $t_{on} = 0.5s$, $t_{off} = 9.5s$, $\alpha_{on} = 0.5s / 10s = 0.05$, $\alpha_{off} = 9.5s / 10s = 0.95$, $I_{on} = 40mA$:

For **sleep mode:**

$$I_{avg} = (\alpha_{off} * I_{avg\_sleep}) + (\alpha_{on} * I_{on}) = (0.95 * 1.3mA) + (0.05 * 40mA) = 3.235\ mA$$

2850mAh / (3.235 mA) / 24 hours a day = **36 days**

Doing the same calculation for **dead/alive mode:**

$$I_{avg} = (\alpha_{off} * I_{avg\_sleep}) + (\alpha_{on} * I_{on}) = 0 + (0.05 * 40mA) = 2.00\ mA$$

Battery Life = 2850mAh / (2.00 mA) / 24 hours a day = **60 days**

For $t_{on} = 1s$, $t_{off} = 1$ hour, $\alpha_{on} = 1s / 3600s = 2.7778 * 10^{-4}$, $\alpha_{off} = 0.99722$, $I_{on} = 40mA$:

For **sleep mode:**

$$I_{avg} = (\alpha_{off} * I_{avg\_sleep}) + (\alpha_{on} * I_{on}) = (0.99722 * 1.3mA) + (2.7778 * 10^{-4} * 40mA) = 1.0375\ mA$$

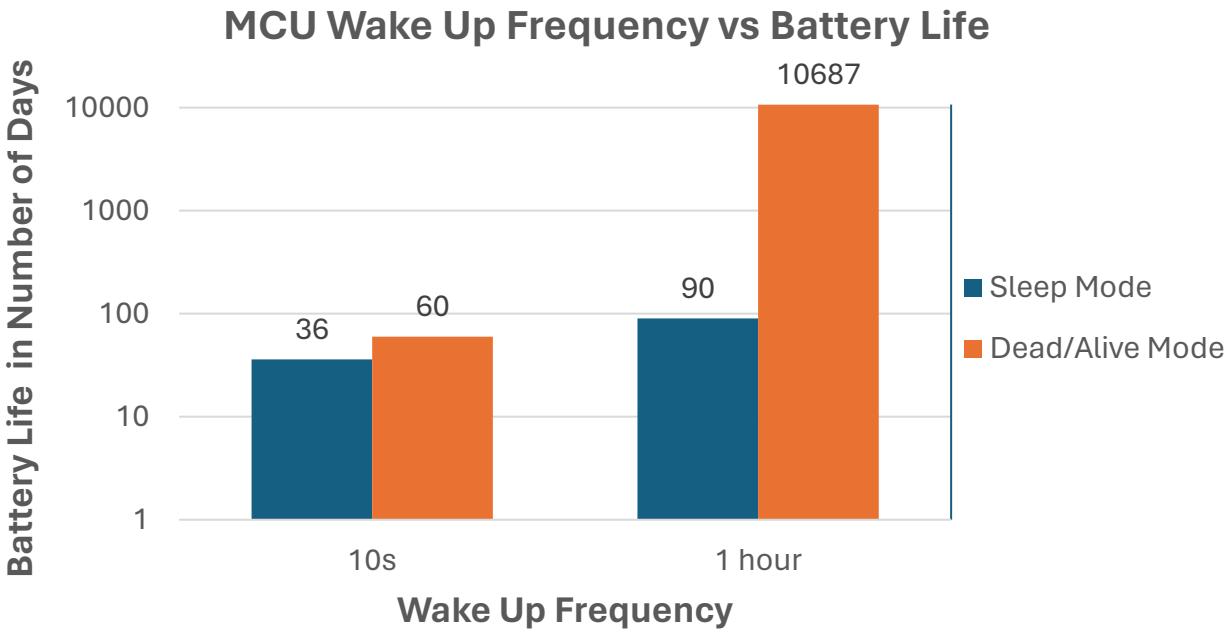
2850mAh / (1.0375 mA) / 24 hours a day = **90 days**

Doing the same calculation for **dead/alive mode:**

$$I_{avg} = (\alpha_{off} * I_{avg\_sleep}) + (\alpha_{on} * I_{on}) = 0 + (2.7778 * 10^{-4} * 40mA) = 0.0111 \text{ mA}$$

Battery Life = 2850mAh / (0.0111 mA) / 24 hours a day = **10687 days = 20 years!**

Here are these numbers summarized in a graph:



As seen by the calculations, dead/alive mode saves significant power. If the microcontroller wakes up every 10 seconds, dead/alive mode gets 60 days on battery vs. sleep mode's 36 days. This disparity significantly widens with slower frequencies where sleep mode only lasts 91 days while dead/alive mode can last nearly forever, or until the battery dies. This is because of the constant 1.3mA of current draw in sleep mode versus in dead mode the microcontroller consumes nearly zero power. Therefore, the longer time the microcontroller stays off, the more power it saves in dead/alive mode compared to sleep/wake mode.

The Latching Circuit

Before delving into the latching circuit, it is important to understand how the Raspberry Pi Pico can be powered on. The Raspberry Pi Pico can be powered on by supplying a voltage source into the VSYS pin of the Pico, located at the 2nd to top rightmost pin. We add a diode in between to prevent either supply from back-powering the other.

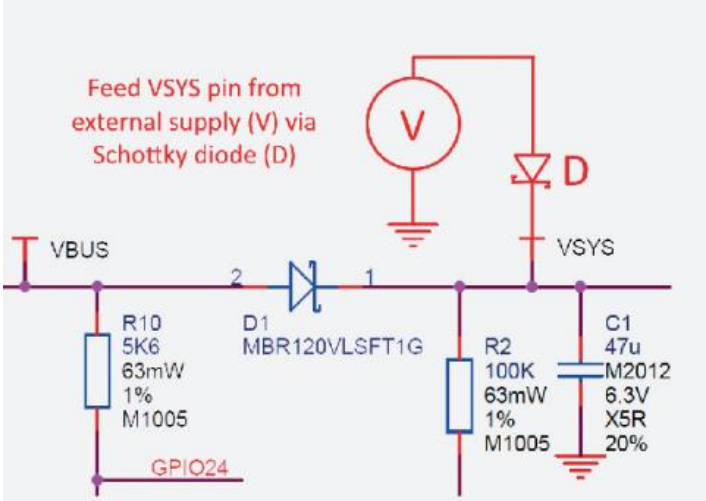


Figure 3: Powering Pico from VSYS, Raspberry Pi Pico Datasheet

Therefore, our latching circuit should act as an enable switch between the battery supply and the VSYS PIN. The following circuit below accomplishes just that.

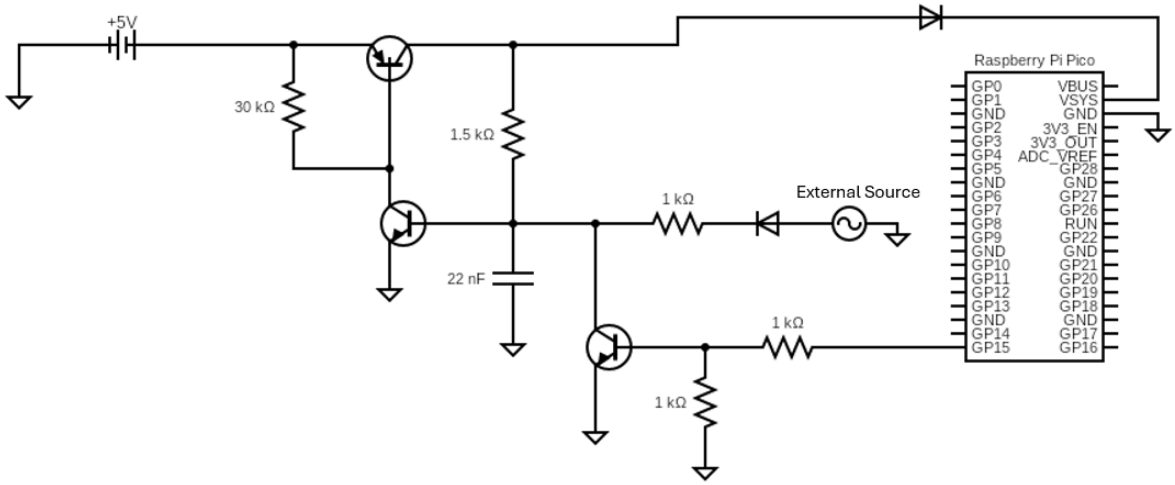


Figure 4: Latching Circuit in Off State

Using the idea from Electronoobs's YouTube video [3], the figure above shows the latching circuit in the latched off or dead state. We can see that the PNP transistor acts as a switch that controls the flow of current from the battery to the VSYS pin of the Raspberry Pi Pico. In the latched off state, the base of the PNP is high, therefore there is no current flowing from the 5V battery to the VSYS pin, hence no power is being consumed.

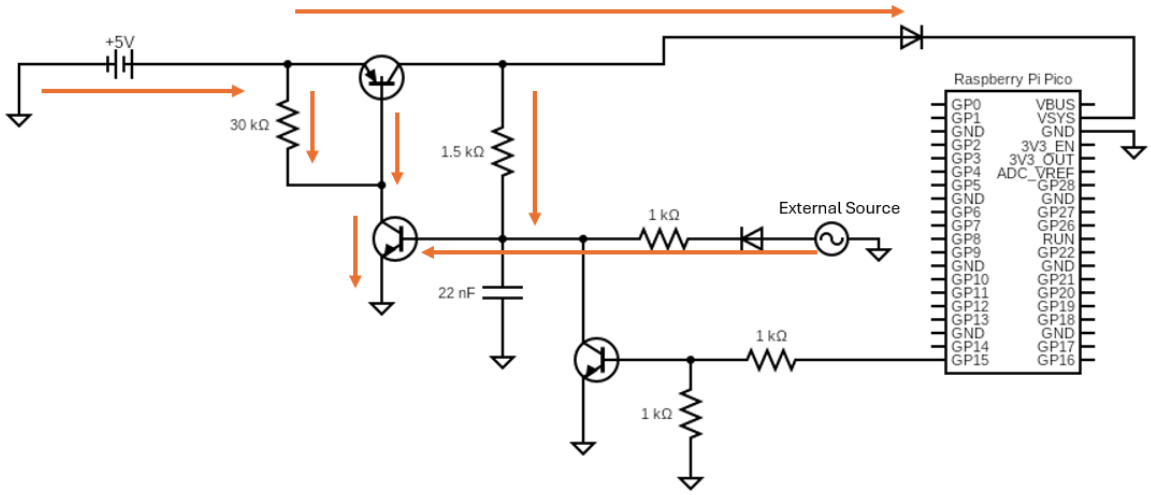


Figure 5: Latching Circuit Toggled On

This circuit quickly changes when an external source is triggered, which will latch the circuit on. Let the red lines in Figure 5 show the direction of current flowing in the circuit. When the external source is triggered, this creates a voltage and current that flows into the base of the NPN transistor, turning it on. Turning the NPN on allows current to flow from the collector to emitter (to ground), discharging the base node of the PNP transistor. This would also turn on the PNP transistor, allowing the battery to supply power to the Pico.

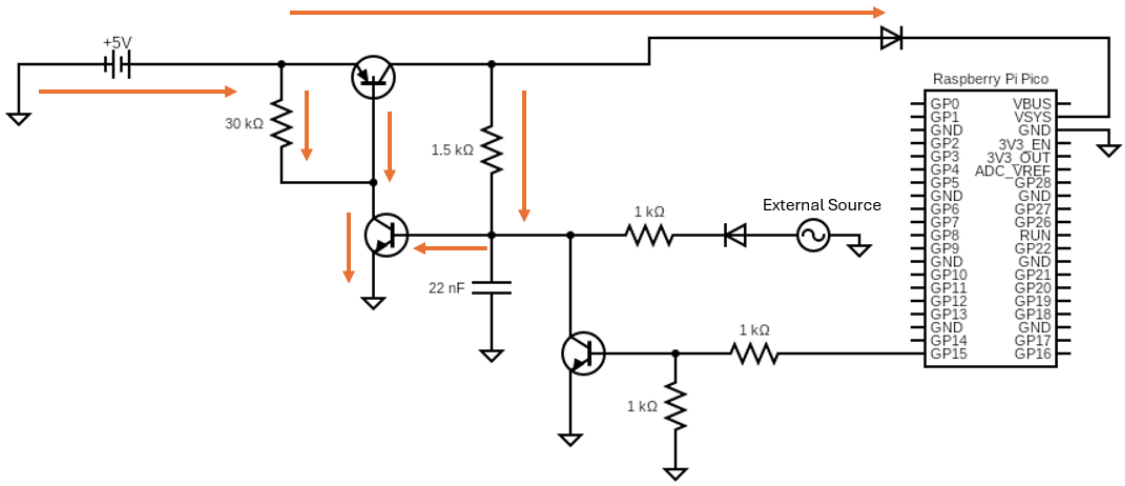


Figure 6: Latching Circuit Latched-On

As the name suggests, the latching circuit stays latched on even when the external current source is off. Once the circuit is toggled on, current is fed back from the PNP's collector through the 1.5kΩ resistor into the base of the NPN transistor, which also has a capacitor to store charge. This would keep the NPN transistor on, keeping the circuit latched on.

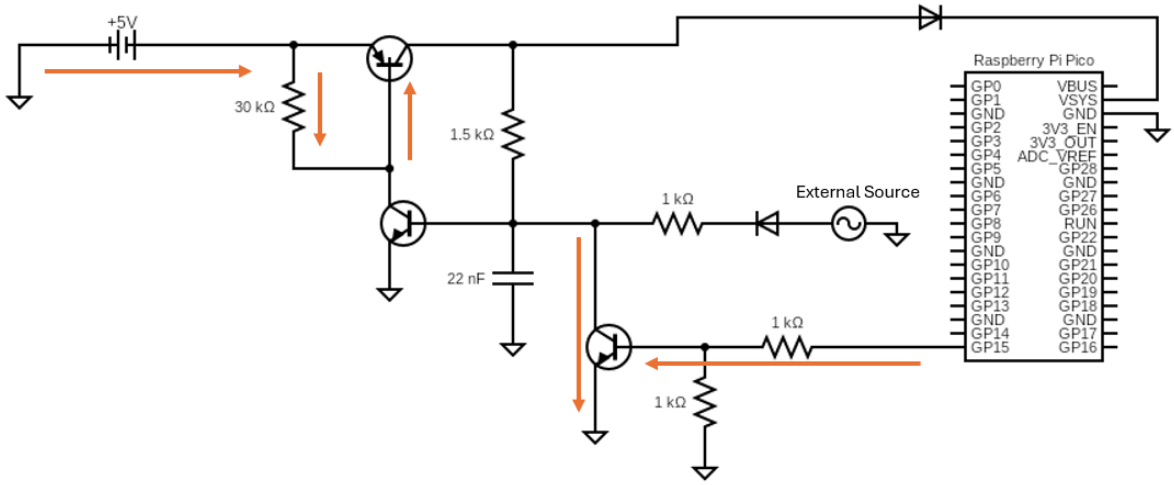


Figure 7: Latching Circuit Latched Off

The figure above shows how the microcontroller can latch itself off once turned on. Once the microcontroller has accomplished its task after waking up, it can put itself back to dead mode. This is done by setting a GPIO pin high, which sends current into the bottommost NPN transistor. This discharges the capacitor node at the base of the middle NPN, pulling it to ground.

Turning off this NPN turns off the PNP, which cuts power from the battery to the microcontroller, putting the circuit back to dead mode, as described in Figure 4.

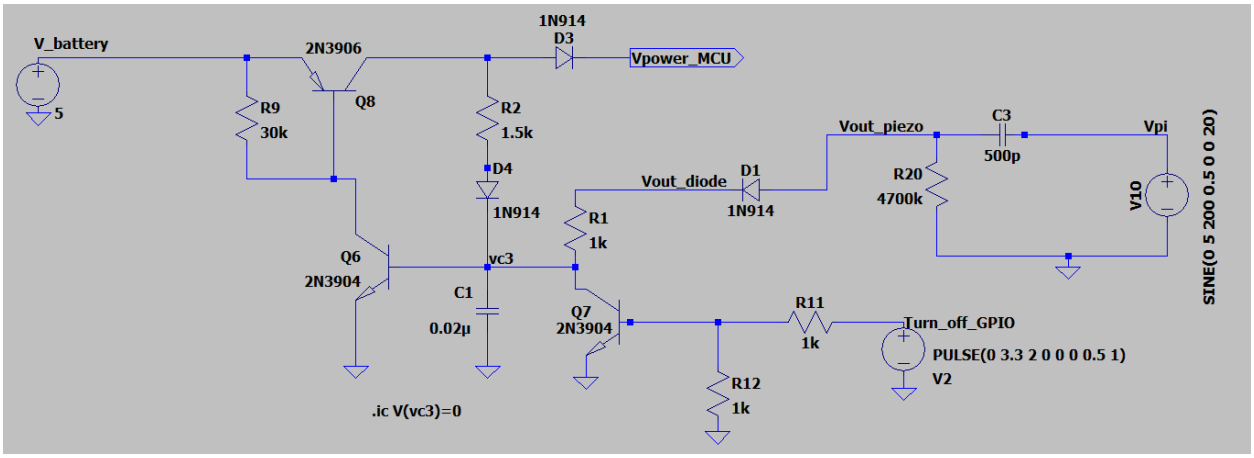


Figure 8: LTSpice Schematic of Latching Circuit

Figure 8 above shows the final modified schematic of the latching circuit. Resistor values of R9 and R2 were chosen through trial and error. The value of R9 did not really affect the circuit, but the value of R2 needed to be low to present a low impedance path from the collector of the PNP to node vc3, the base of the Q6 NPN transistor. To get the latching circuit physically working with the piezo sensor, it was important to understand how a piezo sensor works. The process of integrating the two involved building a simulation in LTSpice that closely models the behavior of the piezo sensor.

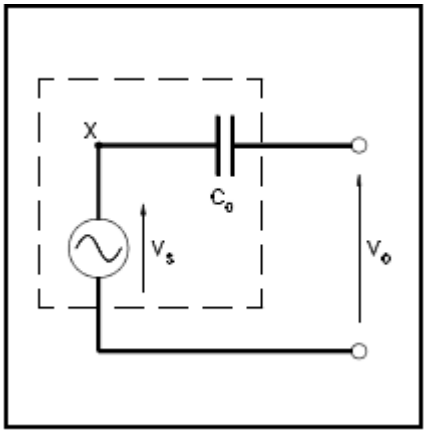


Figure 9: Equivalent Circuit of Piezo film, from Technical Manual, Source [5]

Description	Part No.	Capacitance
LDT0-028K/L	0-1002794-1	500 pF
DT1-028K/L	1-1002908-0	1.3 nF
DT1-052K/L	2-1002908-0	650 pF
DT2-028K/L	1-1003744-0	2.6 nF
DT4-028K/L	1-1002150-0	9 nF
8" x 11" 28 µm	1-1003702-4	30 nF
HYD-CYL-100	0-1001911-1	43 pF

Figure 10: Piezo Film Capacitance, from Technical Manual, Source [5]

Plugging in the sensor did not just immediately work, so this was a process of back-and-forth physical testing and running circuit simulations to understand when something was not working. Following the technical manual shown in Figure 9, the piezo sensor is modeled with an AC voltage source with a series capacitance and a parallel resistance at the output. Note that this creates a CR circuit. The piezo sensor's technical manual estimates the internal capacitance of the LTD0-028K/L to be around 500pF, as seen in Figure 10 above. The piezo sensor's datasheet recommends starting with placing a parallel 1MΩ resistor between the two terminals. Initially with this resistance value, we found that it was hard for the piezo to produce enough voltage to turn on the NPN transistor. Since we wanted the piezo sensor sensitive enough to detect gusts of wind, we found that increasing this to 4.7MΩ allowed the piezo to generate more voltage due to the CR circuit of the piezo film. Next, we put the voltage of the piezo through a diode to rectify the AC voltage produced by the piezo sensor, converting it to DC voltage so that negative voltage swings would not discharge the capacitor in C1. The capacitor value of C1 was chosen to be as small as possible so that a short voltage spike generated by the piezo would be able to charge it up fast as possible, making the circuit more sensitive to the piezo sensor. This same idea was applied when changing R1 from 47kΩ to 1kΩ so that voltage goes through a low impedance path into the base of Q6. Later, we found that despite the open circuit voltage generated by the piezo being sufficient to turn on the NPN transistor at Q6, the actual voltage read at vc3 was not matching that. We found that this was because instead of current flowing into the base of that NPN transistor, current was flowing upwards through the R2 resistor. This was simply solved by placing a diode between R2 and node vc3 to inhibit the unwanted upward current flow.

Now we can discuss what we expect to see from this circuit simulation. At the start, the circuit is latched off. After 0.5 seconds, the piezo sensor is triggered. This is modeled by a 0.5s delay in the AC voltage source. This sends current through the base of NPN transistor Q6, turning it on. This discharges the base node of Q8 and allows Vpower_MCU to get 5V, which turns on the microcontroller. At time = 2s, we simulate the microcontroller setting a GPIO pin high to latch itself off.

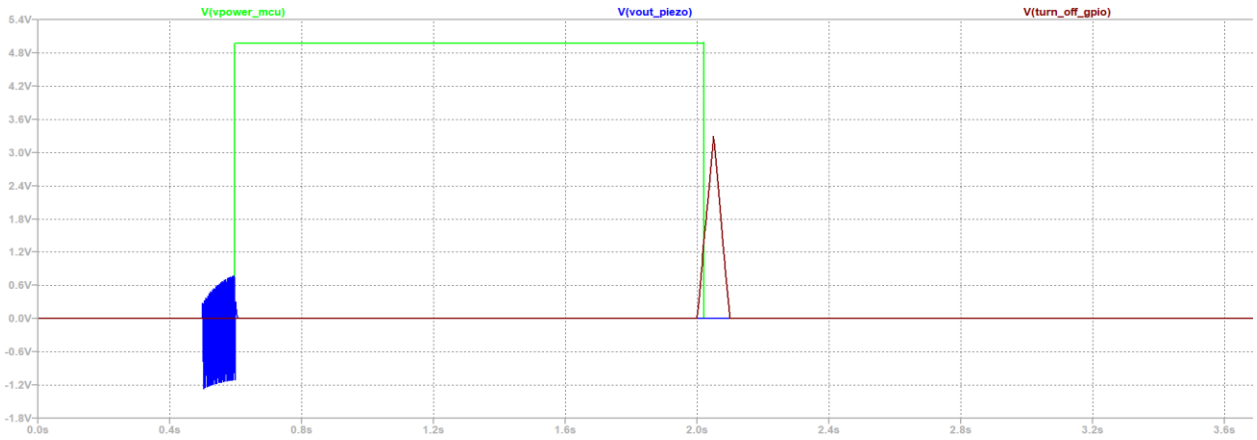


Figure 11: LTSpice Simulated Results of Latching Circuit

The simulated response shows this exact latching behavior. We can see at the start that all signals are low, being the latched off state. At 0.5 seconds, Vout_piezo procduces a short AC signal. In response, Vpower_MCU goes high as there is current flowing from the battery to supply the microcontroller, which would wake the microcontroller up. After a short 1.5-second delay, we see the spike of the GPIO pin being set high, which latches the circuit off, bringing it back to dead mode and consuming almost zero power.

Circuit Implementation with Piezo Sensor, Setting up an Experiment

Before running an experiment, we had to first be able to log time on an SD card. This was accomplished by combining the Pico's RTC program with a FatFs library developed by carlk3 that allows us to write to an SD card via SPI on the Raspberry Pi Pico. Following the instructions on carlk3's GitHub and using Professor Bruce Land's pin connections, I decided to use SPI0 as it was the most optimal given the placement of the SD card adapter. See the steps below for connecting the Raspberry Pi Pico to the SD card adapter.

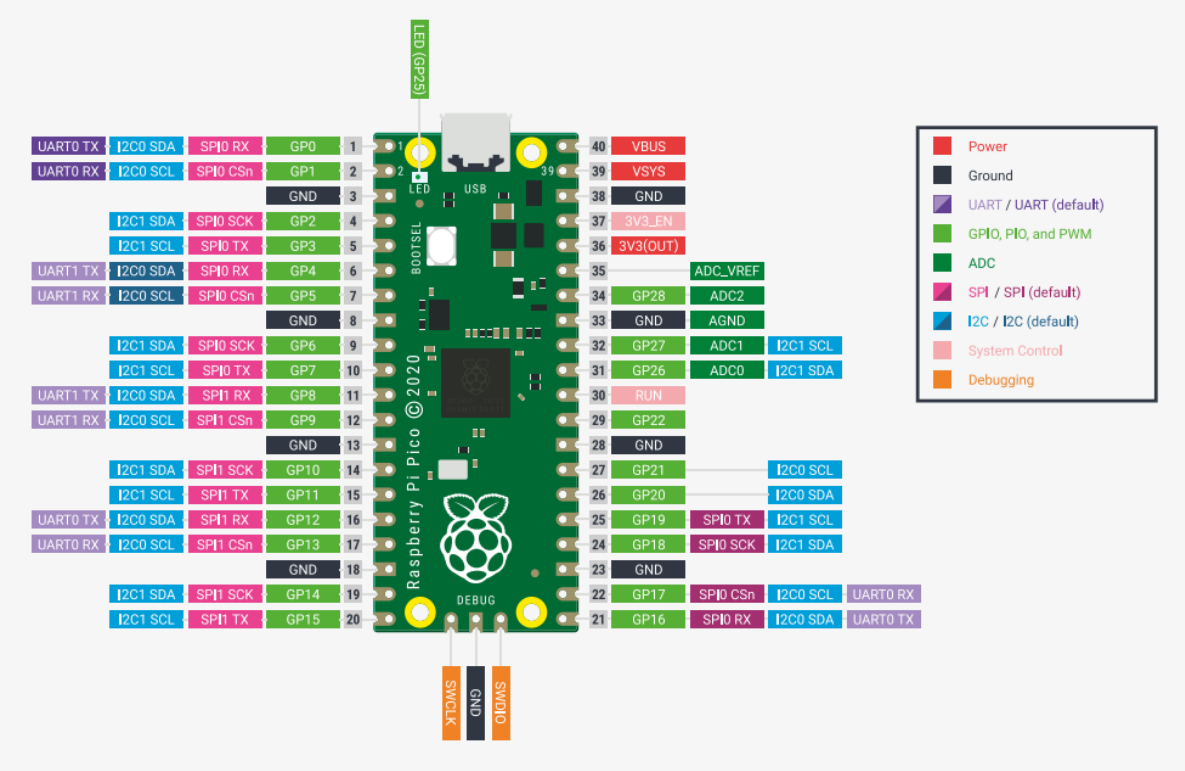


Figure 12: Raspberry Pi Pico Pinout, Source [7]

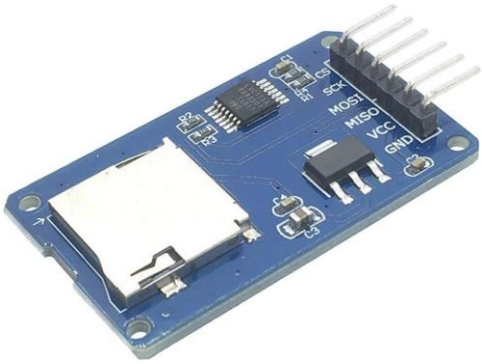


Figure 13: HiLetgo MicroSD Card Adapter

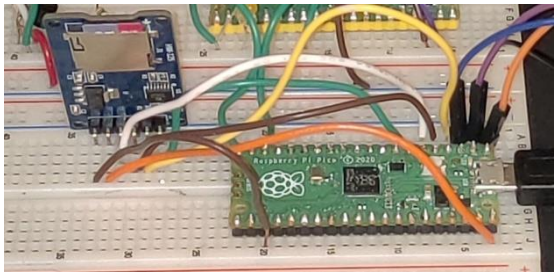


Figure 14: Pico wiring to the SD Card Adapter

The following connections pin connections were made:

- microSD CS --> GPIO 5 SPI0_CS
- microSD SCK --> GPIO 2 SPI0_SCK
- microSD MOSI --> GPIO 3 SPI0_TX
- microSD MISO--> GPIO 4 SPI0_RX
- microSD VCC --> VBUS on PICO (Pin 40)
- microSD GND --> PICO GND (Pin 23)

After these connections were made, a few more modifications were made to carlk3's demo program to get the Raspberry Pi Pico to write to an SD card. In the hardware config file, it was necessary to disable the card detect feature. Next, for some reason, a 20MHz SPI baud rate did not work, so this was lowered to 5MHz in my testing. Once I was able to write to an SD card, I combined this program with an RTC to produce a program that was able to log what time of day a GPIO pin was read high. Lastly, it was time to deploy our experiment!

The figure below shows the initial setup of the experiment.

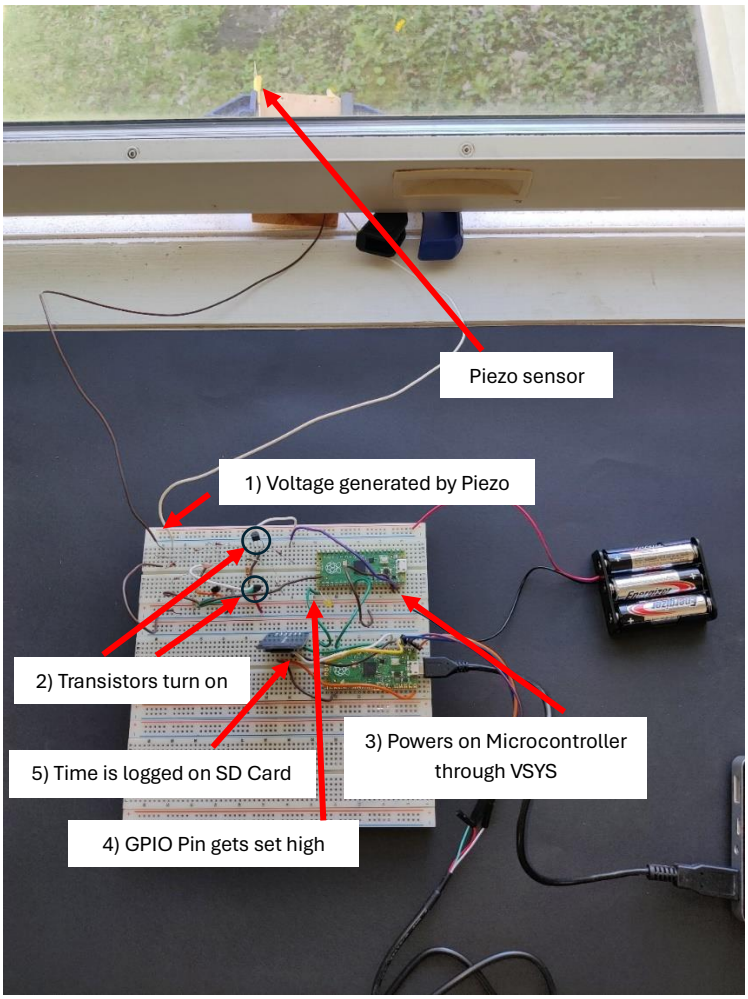


Figure 15: Initial Experiment Setup

We created a 4.5V supply for the Pico by putting three AA batteries in series (3 x 1.5V). The second Pico was plugged into a portable charger and kept permanently running with a program that would log the time the Piezo film was clamped to a wooden block and slipped through an open window crack in an attempt to detect wind breezes to wake up our microcontroller. However, after running for a whole night, we found that the piezo sensor still was not sensitive enough to pick up wind gusts.

Therefore, we decided to attach a small piece of wood flap to add mass and surface area to the tip of the piezo sensor, acting as a sail and increasing its baseline sensitivity. This correlates with the experimental results from the datasheet, as seen below. Also see the final modified setup.

TABLE 1: LDT0 as Vibration Sensor (see Fig 1)

Added Mass	Baseline Sensitivity	Sensitivity at Resonance	Resonant Frequency	+3 Db Frequency
0	50 mV/g	1.4 V/g	180 Hz	90 Hz
1	200 mV/g	4 V/g	90 Hz	45 Hz
2	400 mV/g	8 V/g	60 Hz	30 Hz
3	800 mV/g	16 V/g	40 Hz	20 Hz

Figure 16: Effect of Adding Mass to Piezo Sensor, Source [6]

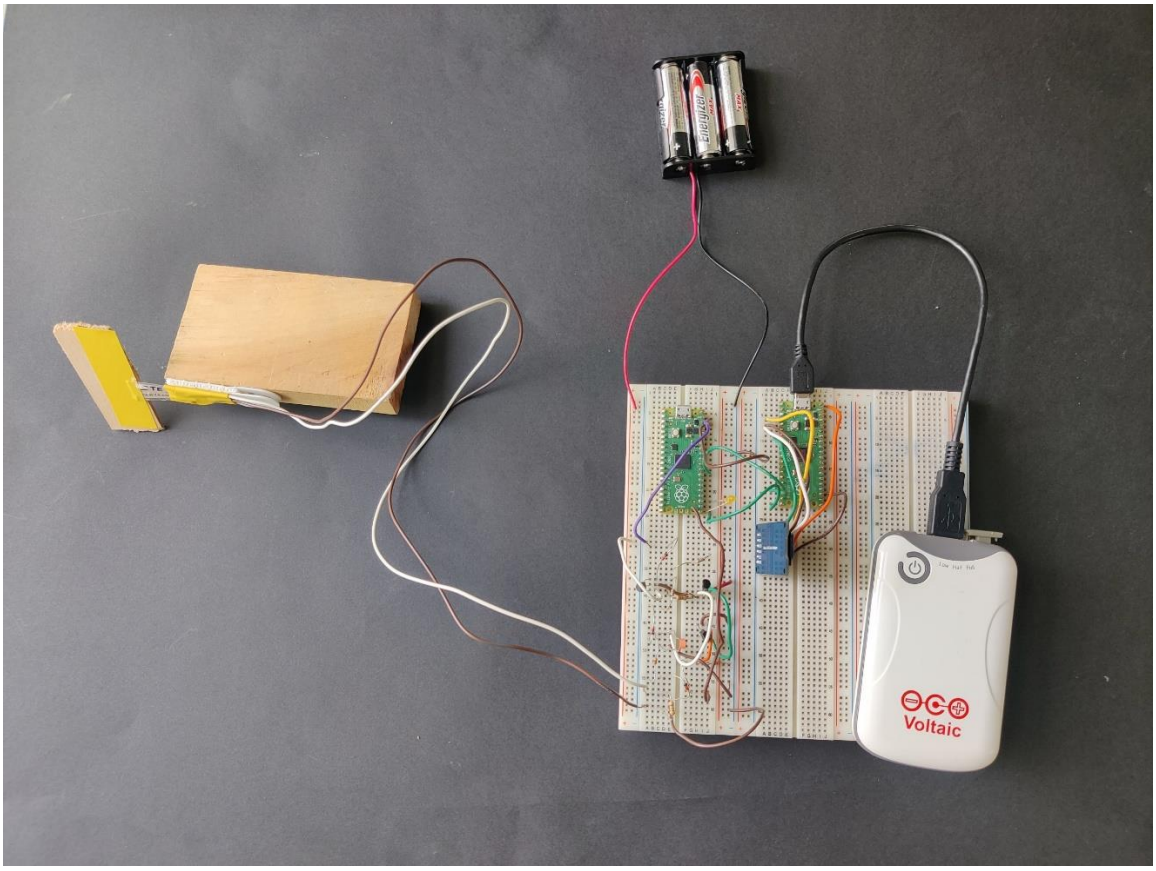


Figure 17: Modified Experiment Setup

Figure 18: Final Experiment Setup, Enclosed

The final experiment encloses the breadboard circuit in a box with the piezo sensor block attached to the top of the box. The box can then be placed anywhere in nature to detect vibration movements.

Experiment on May 2, 2024, in Ithaca, NY

Here are the results after placing the box outside Phillips Hall on Cornell campus for 10 hours.

```
Pressed Thursday 2 May 14:47:52 2024
Pressed Thursday 2 May 14:47:53 2024
Pressed Thursday 2 May 14:47:53 2024
Pressed Thursday 2 May 14:47:54 2024
Pressed Thursday 2 May 14:47:54 2024
Pressed Thursday 2 May 14:47:54 2024
Pressed Thursday 2 May 14:47:54 2024
Pressed Thursday 2 May 14:47:55 2024
Pressed Thursday 2 May 14:47:56 2024
Pressed Thursday 2 May 14:47:56 2024
Pressed Thursday 2 May 15:10:02 2024
Pressed Thursday 2 May 15:10:03 2024
Pressed Thursday 2 May 21:10:27 2024
Pressed Thursday 2 May 21:10:28 2024
Pressed Thursday 2 May 21:10:28 2024
Pressed Thursday 2 May 21:10:28 2024
Pressed Thursday 2 May 21:10:28 2024
Pressed Thursday 2 May 21:10:29 2024
Pressed Thursday 2 May 21:10:29 2024
Pressed Thursday 2 May 21:10:29 2024
Pressed Thursday 2 May 22:32:08 2024
Pressed Thursday 2 May 22:32:08 2024
Pressed Thursday 2 May 22:32:08 2024
Pressed Thursday 2 May 22:32:08 2024
Pressed Thursday 2 May 22:32:08 2024
```

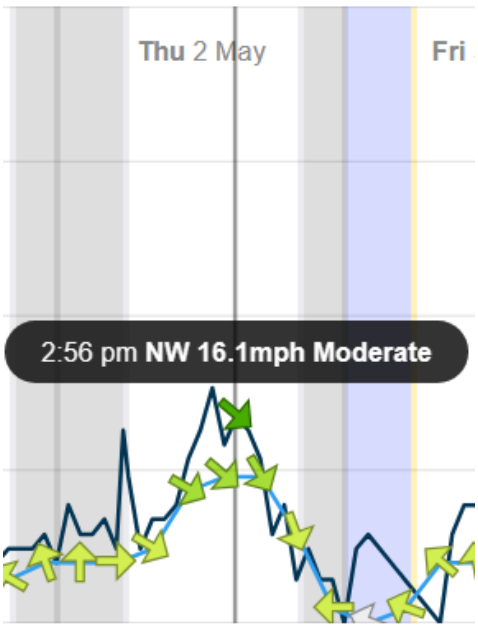


Figure 19: SD Card Log File

Figure 20: Wind Forecast in Ithaca, May 2

As we see from the log file, in the span of 10 hours, the microcontroller naturally woke up four times: at 2:47 pm, 3:10 pm, 9:10 pm, and 10:32 pm. Figure 20 shows the wind forecast on May 2nd. We can see the correlation here because wind speeds spiked highest at 3 pm, which is exactly between the two times the microcontroller woke up. The other two triggers in the nighttime were likely caused by a bird or squirrel poking at the box. Evidently, we can say that this was a successful experiment as natural events were able to trigger the piezo sensor to wake up our microcontroller, log the time of the natural trigger, and the microcontroller was able to latch itself off for minimal power consumption.

Future Work

With two semesters in this project, I was able to demonstrate a working natural timer with the use of a piezoelectric sensor. Future work can include using other external current sources that use natural processes to generate electrical energy to trigger the microcontroller wake-up. An example of this includes a photovoltaic solar cell that converts sunlight into electrical energy. With this, one can measure how often the sun shines at a certain location with minimal power consumption. Another example of this is using a thermoelectric generator that generates electrical power from temperature differences. Thus, one can measure how often temperature fluctuations occur in a certain environment. Future work for this project would also most certainly include a Wi-Fi or Bluetooth connection to communicate to a computer once the Pico wakes up. Currently, the Pico just sets a GPIO pin high when it wakes up, and another Pico program polls that GPIO pin to log the time of the trigger. In the real world, we want to use only one microcontroller and have that microcontroller communicate to a computer server when it woke up from a natural event. The computer server would then log the time of the MCU wake-up. To add Wi-Fi or Bluetooth connection, this would utilize the Raspberry Pi Pico-W, instead of the base Pico microcontroller.

Conclusion

To conclude, the use of natural timers proves to be a way of enhancing energy efficiency in IoT applications. This project addressed the limitations of traditional sleep/wake cycles in IoT devices and created a dead/alive mode powered by natural triggers. With the design and implementation of a piezo sensor with a latching circuit, we were able to demonstrate the feasibility of using natural events, such as wind gusts or animal interactions as triggers for a microcontroller wake-up. The outdoor experiment validated the efficacy of the natural timer system and showcased that the microcontroller could be awakened by environmental stimuli and efficiently latched off to conserve power. Further work on this project could explore integrating other natural energy sources and incorporating wireless communication to a server for logging the time of an event. I would like to thank Dr. Van Hunter Adams for guidance and mentorship in this project that pushed my engineering skills to explore an impactful advancement in IoT applications.

References

[1] Adams, V. H. Power Management in IoT Systems. https://vanhunteradams.com/Talks/IoT_Energy.pdf

[2] carlk3. (n.d.). *Carlk3/no-os-fatfs-SD-SPI-rpi-pico: A fat filesystem with SPI driver for SD card on Raspberry Pi Pico*. GitHub. https://github.com/carlk3/no-OS-FatFS-SD-SPI-RPi-Pico

[3] Electronoobs. (2021, April 25). *Latch Circuit - wake up + 0 power consumption (useful circuit)*. YouTube. https://www.youtube.com/watch?v=Er8fSoeaZD0

[4] Land, B. (2024, April 1). microSD card for Pi Pico RP2040. https://people.ece.cornell.edu/land/courses/ece4760/RP2040/C_SDK_memory/SD_card/index_sd_card.html

[5] Measurement Specialties Inc. (1999, April 2). *Piezo Film Sensors Technical Manual*. Sparkfun. https://www.sparkfun.com/datasheets/Sensors/Flex/MSI-techman.pdf

[6] Measurement Specialties. (2008, October 13). *LDT with Crimps Vibration Sensor/Switch*. Sparkfun. https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/LDT_Series.pdf

[7] Raspberry Pi. (2021, January 21). Raspberry Pi Pico Datasheet. Raspberry Pi Datasheets Retrieved November 15, 2023, from https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf