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

Michael Awad

MEng Field Advisor: Dr. Van Hunter Adams

Degree Date: January 2024

Abstract

Master of Engineering Program

School of Electrical and Computer Engineering

Cornell University

Design Project Report

Project Title: Natural Timers for IoT

Author: Michael Awad

Abstract: This project investigates the merits of replacing timer peripherals with natural triggers in certain embedded systems applications. In some applications, one cares more about battery longevity than regularity of measurement timing. For these applications, a timer-driven sleep/wake cycle might be replaced with an event-driven dead/alive cycle. The embedded system could turn itself completely off and trust a periodic natural process (wind gusts, bird visitation, cosmic rays, etc.) to generate an event which wakes it for a measurement. Such an architecture could improve battery longevity at the cost of timing guarantees for measurements. This project describes the design and testing of a latching circuit which facilitates integration of these "natural triggers" into an embedded system, and compares power consumption between this architecture and the more conventional timer-driven architecture.

Individual Contributions

For the following project, I, Michael Awad, worked with Chris Yang. My individual contributions included building and debugging the latching circuit, writing and debugging code to test the sleep mode and dormant mode of the Raspberry Pi Pico, gathering current consumption data from the Power Profiler II, analyzing the data, debugging the LTSpice simulation, and setting up the solar cell.

Executive Summary

Most microcontrollers offer a low power mode in which all but a small subset of subsystems and peripherals are powered off. This small subset may include a timer or real time clock which wakes the system at a specified future time, or it may alternatively or additionally include General Purpose Input/Output (GPIO) hardware to wake the system by means of an interrupt. Every subsystem which remains on in a low-power mode costs power and, thus, battery life. This project investigates the utility of a latching circuit in some low-power applications. The latching circuit allows for every subsystem and peripheral in the microcontroller to be powered off, which saves power but surrenders guarantees with regard to when the system will wake again by trusting a periodic natural process (wind, a cosmic ray, a bird, etc.) to do the waking. For some applications, this is a reasonable trade.

This project uses the RP2040 as a case-study microcontroller. Though lower-power microcontrollers exist, the RP2040 serves as a reasonable proof-of-concept for comparing the relative merits of timer-based sleep/wake cycles and natural-trigger-based dead/alive cycles in embedded systems applications. Like many other microcontrollers, the RP2040 offers low-power modes. Its datasheet refers to these modes as “sleep” and “dormant.” The latching circuit that has been designed introduces a new low-power “dead mode” that the RP2040 and other microcontrollers can utilize. The latching circuit kills the microcontroller completely, reducing power consumption in “dead mode” to near 0 Watts. The system trusts a natural process to periodically wake it from this mode.

This project involved designing and engineering a system that allows for the investigation of the merits of replacing timers with natural triggers in certain embedded systems applications. To that end, the engineered system has certain properties. The microcontroller can successfully move to the latched-on state by using some external source such as a solar cell, it can be latched-off using a GPIO pin that is controlled by the same microcontroller, and lastly, infrastructure has been developed for comparing the average power consumption of the naturally triggered system to that of the timer-driven system.

Introduction

If I were to ask a farmer how often a bird is visiting their birdfeeder, the farmer would most likely respond with every ten minutes. The question then becomes, does the farmer mean *exactly* every ten minutes, or on the order of tens of minutes? It is more than likely that they mean on the order of tens of minutes. In this instance, it is fair to say that the farmer is insensitive to the exact timing. With this in mind, we can use a naturally periodic event to calculate how many birds are visiting the bird feeder.

The beginning of this process was to investigate the sleep mode of the Raspberry Pi Pico. The datasheet reports expected current draw for each low power mode, and the first step of this project was to confirm these values experimentally. In order to do so, I used the same program that the datasheet used and put the microcontroller in sleep mode, and I used the Power Profiler II to measure the current flowing through the Pico. After running this experiment and confirming the data was correct, I could proceed with the project.

The first step in this was to create the circuit and simulate it on LTSpice in order to confirm the hypothesis that we can latch on and off the microcontroller. The results proved that we can both latch on and off the circuit as expected. This allowed me to begin building the circuit.

The circuit building process was incremental. I broke up the circuit into three separate sections: elements that keep the circuit latched-on, elements that keep the circuit latched-off, and the sources of current. I built each part of the circuit separately to ensure proper functionality. Once each was unit tested, I integrated each part of the circuit together.

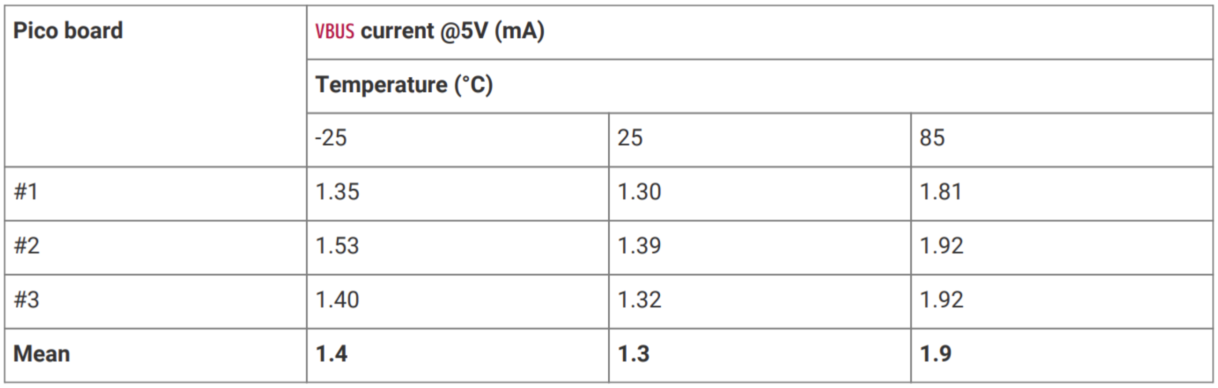
Finally, I compared power consumption between timer-driven sleep/wake mode and naturally triggered dead/alive mode. The Pico would be in sleep mode or dead mode for the same amount of time, wake up and turn on an LED, wait five seconds, and go back to either dead or sleep mode. The data show over a 50% reduction in power consumption when using the dead/alive mode versus the sleep/awake mode.

Background

A technique utilized by embedded system programmers to save power is an implementation of a sleep/wake mode.It is only put into these modes in order to keep the microcontroller's real-time clock on in order to know when it should complete the next task. This project uses the RP2040 as a case-study microcontroller. Though lower-power microcontrollers exist, the RP2040 serves as a reasonable proof-of-concept for comparing the relative merits of timer-based sleep/wake cycles and natural-trigger-based dead/alive cycles in embedded systems applications. Since we are event focused in some situations, why can't we use an event to turn on the microcontroller in order to avoid using a sleep mode? We replace sleep mode with *dead* mode, in which all subsystems in the device are powered off. The following report will go into detail about using a latching circuit that is latched-on by an external current source that can be controlled by nature. The microcontroller then would turn itself off by sending current through another part of the latching circuit, which as a result would latch the microcontroller off putting it into *dead* mode.

Investigation of Sleep Mode of Raspberry Pi Pico

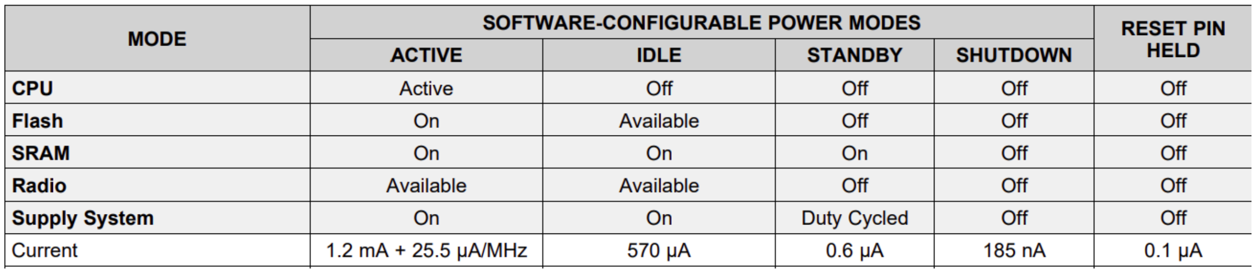
The first task of this project required experimental confirmation of the sleep-mode current consumption that the datasheet reports. I gathered these measurements with a Nordic Power Profiler Kit II, which can measure currents as small as pico-amps. The Raspberry Pi Pico's datasheet included a table that included the average current measure whilst the Pico was in sleep mode. A figure of that table is included 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

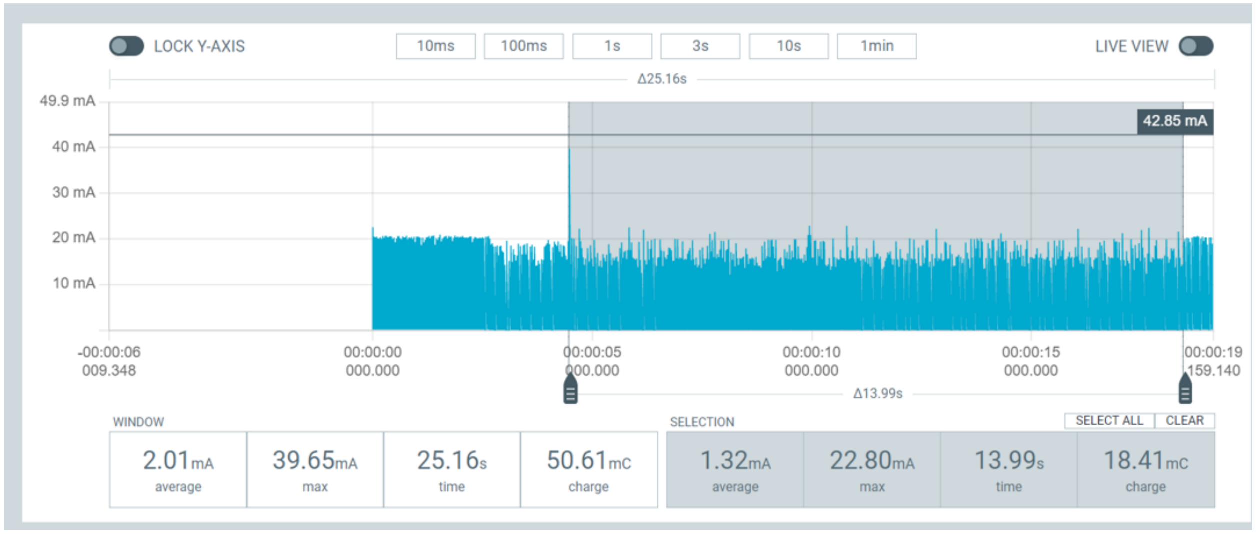
This table ultimately concludes that while in sleep mode, there is 1.3 mA of current flowing. This is significantly higher than most microcontrollers that are designed to optimize their low-power modes. For comparison, let's look at the CC1310 — a low-power microcontroller designed by Texas Instruments. The datasheet for the CC1310 contains a similar table that shows the expected current when running in different power modes. In STANDBY mode, which is the closest comparator to sleep mode in the Raspberry Pi Pico, the average current that should be flowing through the microcontroller is only 0.6 microamps (see Figure 2). To compare, the Raspberry Pi Pico's current in sleep mode is more than 2100 times *larger* than the STANDBY mode in the CC1310. This clearly shows that again, if low-power consumption is something desired in a design, the Raspberry Pi Pico cannot be considered as of now.



MODE	SOFTWARE-CONFIGURABLE POWER MODES				RESET PIN HELD
	ACTIVE	IDLE	STANDBY	SHUTDOWN	
CPU	Active	Off	Off	Off	Off
Flash	On	Available	Off	Off	Off
SRAM	On	On	On	Off	Off
Radio	Available	Available	Off	Off	Off
Supply System	On	On	Duty Cycled	Off	Off
Current	1.2 mA + 25.5 µA/MHz	570 µA	0.6 µA	185 nA	0.1 µA

Figure 2: CC1310 Power Mode Data

The next steps were to ensure that the data that the Pico's datasheet was providing was accurate. The data that the datasheet provided was based on the running code that was given in the Pico-SDK, *hello_sleep.c*. This program would use the real-time clock (RTC) to set an alarm for 15 seconds. Until that alarm was triggered, the Pico would remain in sleep mode. To gather my data, I ran *hello_sleep.c* and using the power profiler, found the average current flowing through the Pico.



WINDOW

2.01mA	39.65mA	25.16s	50.61mC
average	max	time	charge

SELECTION

1.32mA	22.80mA	13.99s	18.41mC
average	max	time	charge

Figure 3: Average Current Flowing Through Pico During hello_sleep.c

As depicted in Figure 3, the average current flowing through the Pico while it is in sleep mode was 1.32 milliamps. This value is within two percent of the expected value, which confirmed that the value in the datasheet was accurate.

Latching Circuit Theoretical

After experimentally confirming power consumption in sleep mode, the next step was to understand the latching circuit. In the following section, I will walk through how the latching circuit should work as a system, and each state individually.

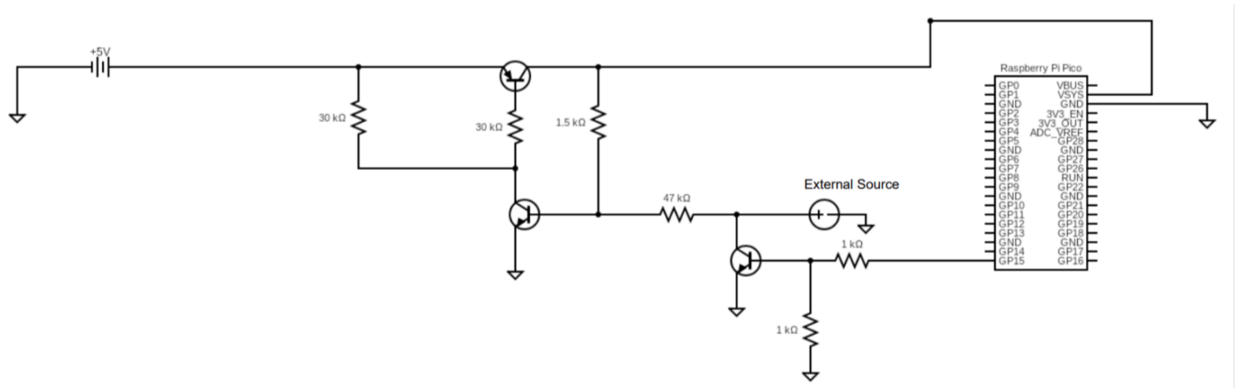


Figure 4: Latching Circuit in Off State

The idea behind the latching circuit is that through the external source, we can latch this circuit on which will turn on the microcontroller. Through the GPIO pin that is connected to the base of the bottom transistor, we can kill the microcontroller and latch the circuit off. The PNP transistor that has the collector end connected to the VSYS of the Raspberry Pi Pico acts as a high-side switch. When the NPN transistor is turned on and current can flow through it, that means that there is current flowing into the base of the PNP transistor. Since current is flowing through the base of the PNP transistor, that allows current to flow from the emitter to the collector which would supply the Pico with current to turn on. The above schematic shows that this circuit is in the off state. This is because the external source has not been turned on yet, which means no current is flowing through the NPN transistor. Essentially, this entire circuit is just an open circuit and there is no current flowing at all.

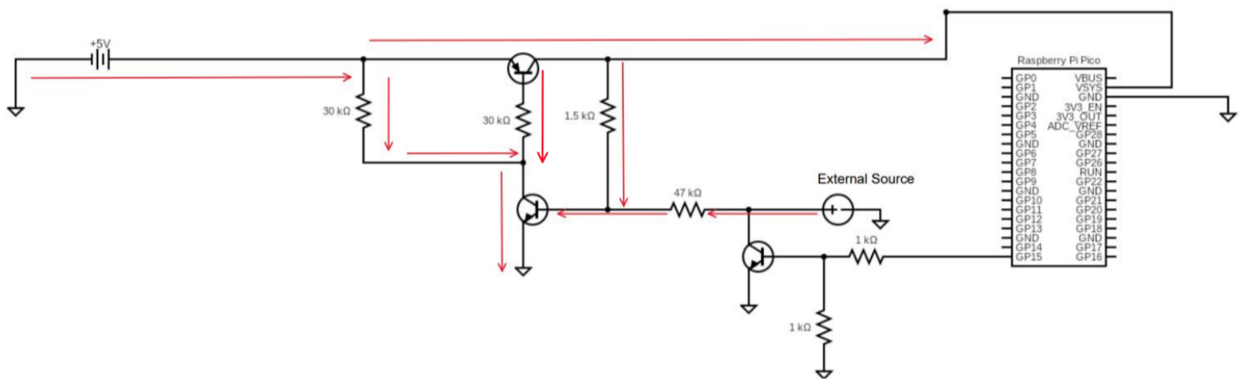


Figure 5: Latching Circuit Toggled On

Let the red lines in Figure 5 show where the current is flowing. The circuit above shows the latching circuit being toggled on. This means that the external source that is sending current into the base of the NPN transistor is triggered. Since the NPN transistor is turned on, current can flow through the emitter end and to ground. This lets current flow through the base of the PNP transistor, which turns that transistor on. Current can then flow from the emitter to the collector, which will turn on the microcontroller. Current also flows through the 1.5k Ohm resistor that will flow back into the base of the transistor.

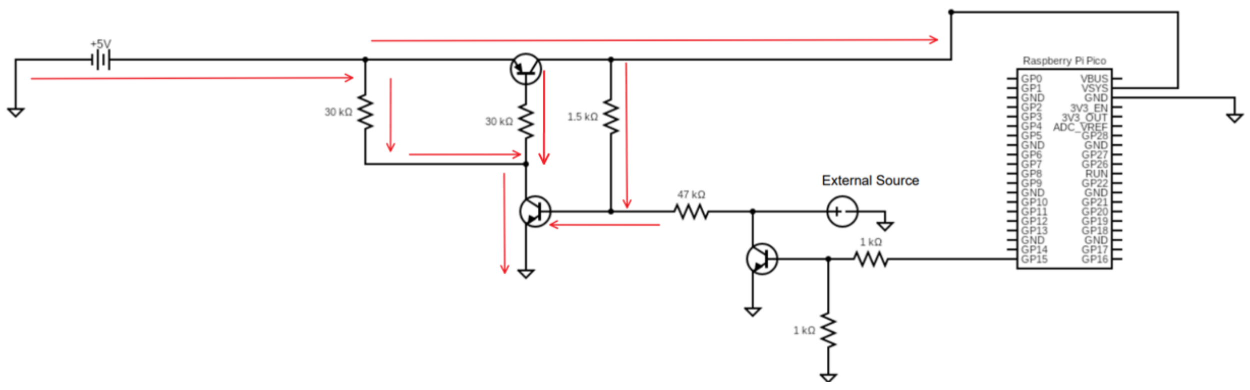


Figure 6: Latching Circuit latched-on

The figure above shows the circuit being latched-on, that is to say, the external current source is off and the microcontroller is still turned on. The current that is flowing through the 1.5k Ohm resistor that flows back into the base of the NPN transistor is what keeps the circuit latched-on. That current is what keeps the NPN transistor turned on which allows current to flow in this loop that can keep the microcontroller turned on.

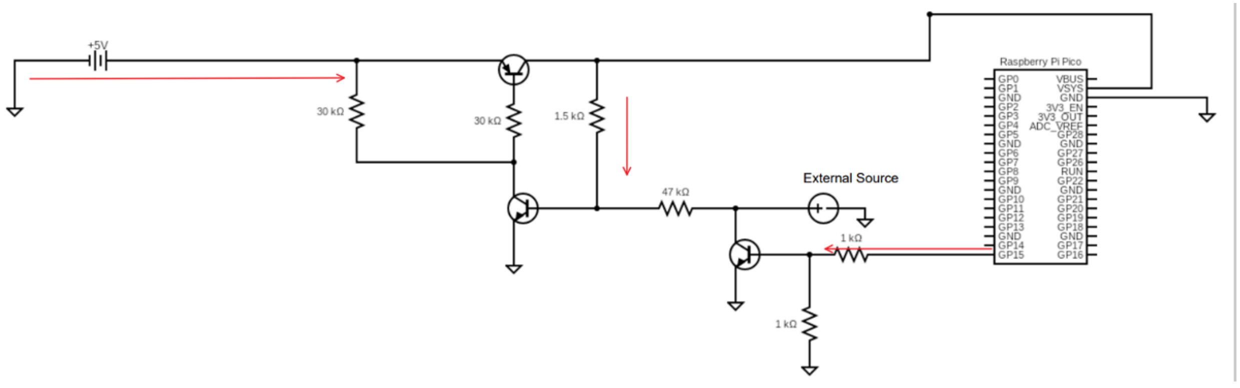


Figure 7: Latching Circuit Toggled Off

The figure above shows how the latching circuit is latched-off. The scenario would be: the microcontroller turned on, did some arbitrary task, and now is ready to go back into dead mode. Once the GPIO is set high, it is connected through a resistor to the base of the bottom right NPN transistor, this pulls down the base of the NPN transistor that is responsible for keeping the PNP transistor on. This circuit would then be put back into the off state that we see in Figure 4.

Latching Circuit Simulation

Now that we are confident in the theory behind the latching circuit, the next step is to build a simulation of the circuit in LTSpice.

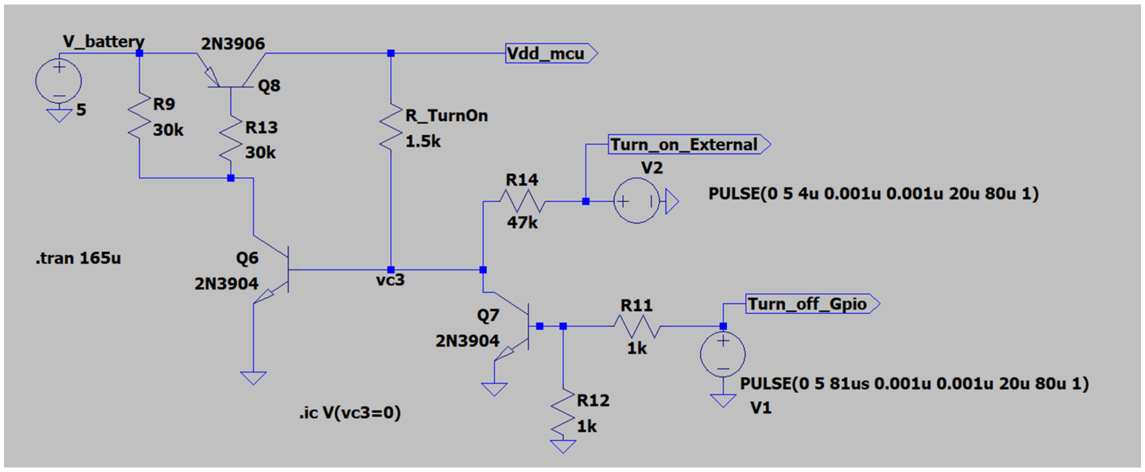


Figure 8: Latching Circuit Created on LTSpice

It is imperative to understand what we should see when running this simulation. First, V2 is what would be sending the external current that would turn on Q6. After a 4-microsecond delay, we send a pulse of 5 Volts through R14. That pulse has a rise and fall time of 0.001 microseconds, and the pulse is high for 20 microseconds. After an 80-microsecond delay, another pulse is sent that has the same timing as the first pulse. This comes from V1, which mimics the GPIO being set high in order to kill the circuit. In order to run this simulation successfully, we need to be aware of what constitutes a successful run. What we should see is that the output Vdd_mcu is high from the moment the V2 pulse is high. Vdd_mcu should stay high until the V1 pulse goes high, then it should turn off. What this would mean in practice, is that the microcontroller is latched-on from the external source, and then it would be turned off using the GPIO. The last thing to add is that we set an initial condition on node vc3 to be 0. The reason behind this is that we want to show what exactly is latching the circuit on. If that node was high initially, we would not be able to definitively say that V2 latched the circuit on. This is also why V2 has a small delay associated with it. This will be able to clearly show that the Vdd_mcu is off until V2 is high.

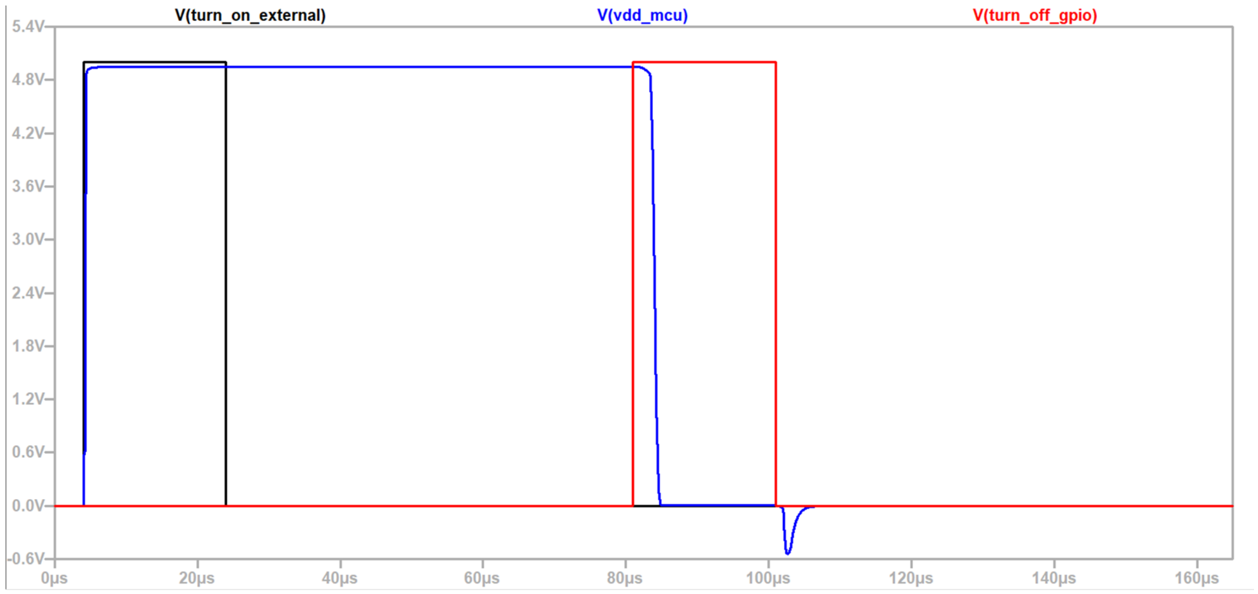


Figure 9: Simulated Response of LTSpice Latching Circuit

The simulated response of the circuit is exactly what we theorized it to be. The black trace is V2 (the external source), the blue trace is the Vdd_mcu output (the microcontroller Vsys pin), and the red trace is the V1 (the GPIO source that is responsible for killing the circuit). This shows that Vdd_mcu is at 0 Volts until V2 is set high. Then, Vdd_mcu is latched-on. We can say that it is latched-on because even when V2 turns off, the Vdd_mcu is still at 5 Volts. That is until V1 goes high. V1, which is acting as the GPIO being set high, immediately turns off the microcontroller. Once all the current is pulled down through the emitter, the microcontroller is killed and we are back in our latched-off state. This successfully demonstrated the latching circuit. Additionally, from the simulation, we are able to draw some early conclusions about this process. Namely, we are able to reduce the power consumption to nearly 0 Watts by turning off the microcontroller.

Experiment: Sleep/Awake Mode vs. Dead/Alive Mode

Now that we have a theoretical understanding and a working simulation, we can now move on to building this circuit and running tests. The objective of this is to find out if going from sleep to awake mode consumes more or less power than going from dead mode to alive mode. It is clear that the microcontroller being off will consume less power than the microcontroller being on. But, the question is does the increase in current that is needed to bring the Pico out of dead mode offset the difference in current?

We want to compare apples to apples in this experiment, that is to say, we want to make sure the microcontroller is doing the same thing in both modes. We also want to make sure that this process is repeated a number of times in order to have a good estimate of the current flowing in both modes. The following flow chart will depict what is happening during the sleep mode experiment.

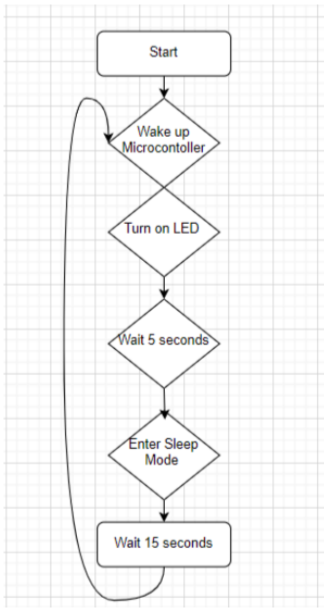


Figure 10: Flow Chart of Sleep Mode Experiment

One important note about this is the wake up microcontroller block. When the Raspberry Pi Pico goes into sleep mode, it has to be awakened with a separate function. That function will be explained in another section. Ultimately, this experiment is just turning on an LED, then going into sleep mode, and then waking up and repeating. The figure below is a flowchart of the dead mode experiment.

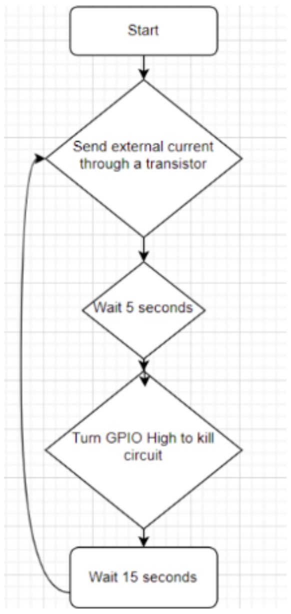
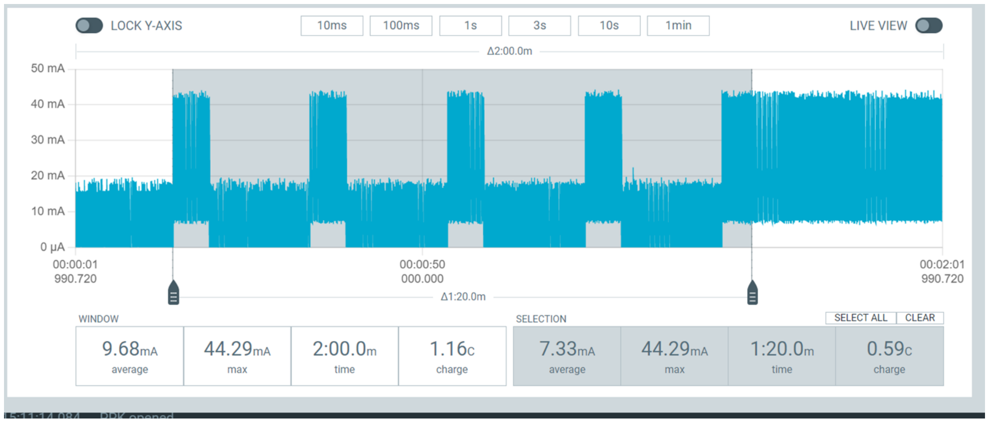

Start
Send external current through a transistor
Wait 5 seconds
Turn GPIO High to kill circuit
Wait 15 seconds

Figure 11: Flow Chart of Dead Mode Experiment

Again, it is important to note what we expect to see during this experiment. The hypothesis here is that we will have less average current when going from dead to alive mode than going from sleep to awake mode. One thing to note is that this experiment is not using a solar cell as the external source. Rather it is using a second Pico board that is setting a GPIO pin high to send current through the base of the NPN transistor. The current was measured with the Power Profiler II and the results will be discussed in the next section.

Results

After running the experiment, the data was gathered and analyzed. The figure below shows the average current going from sleep mode to awake mode.



WINDOW
9.68mA average | 44.29mA max | 2:00.0m time | 1.16C charge

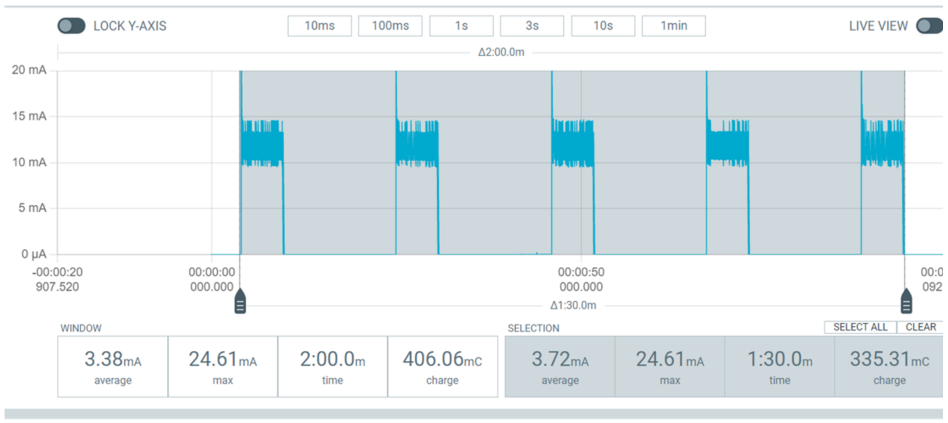
SELECTION
7.33mA average | 44.29mA max | 1:20.0m time | 0.59C charge

Figure 12: Sleep Mode Experimental Data

The plot above shows that the average current going from sleep mode to awake mode (and repeat) is 7.33 milliamps. There are other key notes about this plot that should be mentioned. Namely, the volatility of the current. Even in sleep mode, the Pico can see current spikes up to 20 milliamps which is undesirable. Additionally, when in awake mode, there is still great volatility. The maximum current spike was 44.29 milliamps, which is incredibly high. Considering that the only action that this program is running is turning on an LED and waiting, it is much higher than anticipated. Lastly, to calculate the average power consumption, we will be using the average current. The average power consumption equation and results are below.

$$P_{avg} = V_{in} * I_{avg} = 5V * 7.33mA = 36.65\, mW$$

Figure 13 shows the average current going from dead to alive mode.



WINDOW
3.38mA average | 24.61mA max | 2:00.0m time | 406.06mC charge

SELECTION
3.72mA average | 24.61mA max | 1:30.0m time | 335.31mC charge

Figure 13: Dead to Alive Mode Experimental Data

The plot above shows that the average current going from dead to alive mode (and repeat) is 3.72 milliamps. Some other key notes here: much less volatility in both dead mode and alive mode. This is more desirable in any system, and specifically in low-power mode it bodes well. The average power consumption in the dead and alive experiment is shown below:

$$P_{avg} = V_{in} * I_{avg} = 5V * 3.72mA = 18.6\,mW$$

This shows a reduction in power consumption by over 50 %, which confirms our hypothesis that the dead and alive mode would result in less power consumption. There are additional benefits that come along with low-power consumption. Since the Pico consumes less power, it can be powered with a battery for a longer period of time. Typically, an Alkaline battery can store anywhere from 1700 milliamp-hours (mAh) to 2850 mAh. Since we know the average current, we can calculate how many days each system could be powered on by an alkaline battery. For this calculation, we will say that the average capacity of the battery is 2250 mAh. For the sleep/awake mode, the calculation is as follows.

$$2850\,mAh\,/\,7.33\,mA = 388.81\,hours$$
$$388.81\,hours\,/\,24\,(hours/day) = 16.2\,days$$

This means that the Pico would be able to last *only* 16.2 days using the sleep/awake mode in our current configuration. Comparatively, the dead/alive mode calculation is as follows.

$$2850\,mAh\,/\,3.72\,mA = 766.13\,hours$$
$$766.13\,hours\,/\,24\,(hours/day) = 31.9\,days$$

This means that the Pico would be able to last over double the amount of days using our dead/alive mode method.

Conclusions

Ultimately, this project has reinvigorated my love for engineering. This research has the power to leave engineering in a more affordable and accessible place. There lies a plethora of applications for use of this technology, namely in digital agriculture. At the cost of timing, a farmer can gather data that they need whilst at the same time increasing the battery life of the project.I came into this project with the goal of doing some arbitrary task in order to graduate. I am leaving this project with more curiosity than ever. I am incredibly proud of the work that I have accomplished and what this may mean for embedded engineering. I would also like to extend my gratitude to

Dr. Van Hunter Adams, he quickly became a mentor to me and I believe that he pushed both my research and engineering capabilities beyond what I thought was possible.

References

[1] Texas Instruments. (2021, September 22). CC1310 SimpleLink™ Ultra-Low-Power Sub-1 GHz Wireless MCU. Retrieved November 15, 2023, from

CC1310 SimpleLink™ Ultra-Low-Power Sub-1 GHz Wireless MCU datasheet (Rev. D)

[2] Electronics Tutorials. PNP Transistor Tutorial - The Bipolar PNP Transistor. (n.d.). Retrieved November 15, 2023, from

https://www.electronics-tutorials.ws/transistor/tran_3.html

[3] Nordic Semiconductor. Power Profiler Kit II. (n.d.). Retrieved November 15, 2023, from

https://www.nordicsemi.com/Products/Development-hardware/Power-Profiler-Kit-2

[4] 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