Designing a Low-Cost Universal Chip Programmer Using an FPGA

Jacob Hazelbaker

July 31, 2018

Abstract

Universal chip programmers enable the user to read and write to a wide variety of programmable chips, such as EEPROM chips. However, these devices are often quite costly and the lower-end models often are not capable of interacting with a very wide array of programmable chips. The goal of this research is therefore to design a low-cost universal chip programmer using an FPGA and open-source IP core to readily implement I^2C communication between the programmer and the chip. To safely meet the different voltage and current requirements of a wide variety of programmable chips, a custom PCB has also been designed. The flexibility of the FPGA also allows for potential future development to this project, such as the possibility of adding in logic analyzer capabilities.

1 Introduction

Programmable chips provide a great deal of flexibility in designing an electrical circuit, replacing what otherwise might be a vast array of logic gates with a single chip. Over the years many different types of programmable chips have been developed to meet various electrical engineering needs. The result is a veritable cornucopia of programmable chips with varying voltage and current requirements, differing form-factors, and disparate communication protocols required to perform read and write operations with the programmable chip. For these reasons, many chips require a specific unique type of programming device to be able to safely program and read the chip.

Universal chip programmers, however, strive to be compatible with a variety of different programmable chips. Such devices help negate the need for purchasing multiple different traditional chip programmers to be able to interact with different models of programmable chips. Having a universal chip programmer on hand can thus be exceedingly convenient and could potentially save time by nullifying the need to go out and purchase a new traditional chip programmer when a project requires the use of a different kind of programmable chip.

Even so, the often high cost of universal chip programmers makes it difficult for many engineers to justify the investment. For example, the Xeltek SuperPro 6100 currently supports 102,239 different programmable chips, yet costs $1,595. [2] There are several relatively inexpensive devices on the market which are advertised as universal chip programmers, but unfortunately such devices are by and large only capable of writing to EEPROM chips. Moreover, the universal chip programmers currently on the market offer little opportunity for later expanding its functionality, such as adding in the ability for the universal chip programmer to also serve as a logic analyzer. The design decisions for this project, therefore, sought to create an open-source, relatively low-cost universal chip programmer which could also perform other functions, such as analyzing logic levels.

2 Hardware Selection

To provide a flexible development board which could be programmed to be able to read and write to various chips, it was decided that a Field Programmable Grid Array (FPGA) would be used for this project. The potential use of high-speed block RAM, intrinsic to many FPGAs, is particularly appealing for this project, considering that data will often need to be transferred to and from the chip very quickly.

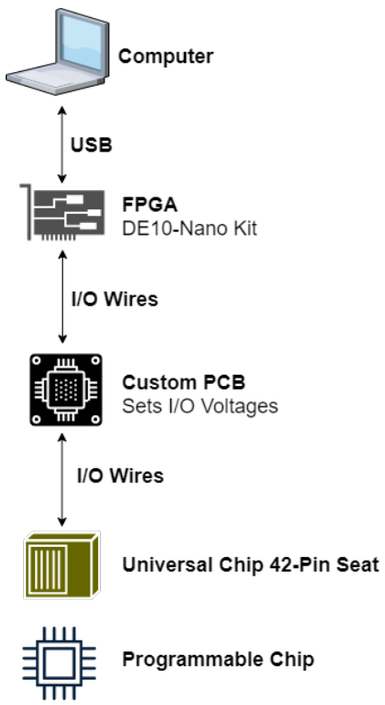


Figure 1: Hardware Block Diagram

The often large amount of I/O pins provided on FPGA boards would also prove to be quite beneficial, making it possible to connect each pin of a programmable chip with its own dedicated I/O pin on the FPGA board. After an extensive search, the DE10 Nano FPGA kit was chosen, which features a Cyclone V FPGA, a dual-core ARM co-processor, and 1 GB of DDR3 RAM. [1]

The FPGA is to be connected to a custom PCB which will ensure that the correct voltage and current levels are applied to the programmable chip. A 42-pin universal chip seat has been selected to provide a convenient way to connect to a wide array of different programmable chips. I/O jumper wire ribbons have been purchased to connect the FPGA to the custom PCB and the custom PCB to the universal 42-pin seat where the programmable chip will reside.

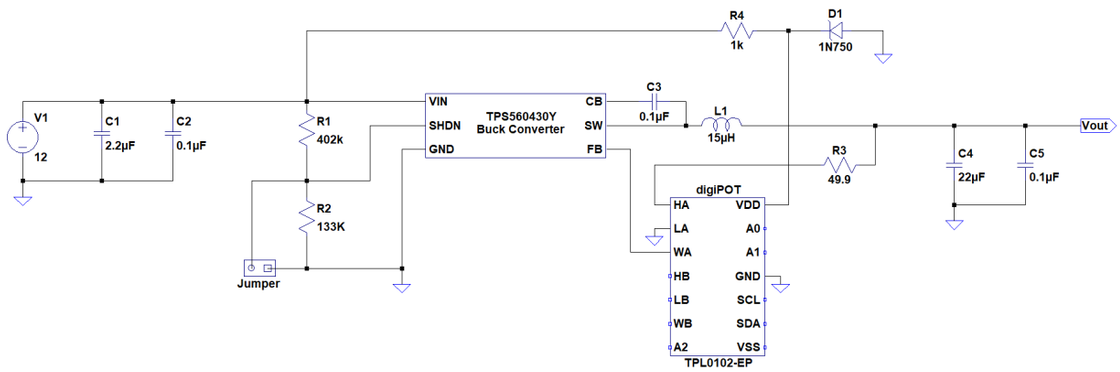


Figure 2: Custom PCB Will Power the Chip Via a Buck Converter Set by a Digital Potentiometer

3 Custom PCB Design

Different brands and models of programmable chips often require that different voltages and currents be used when interacting with the chips. For this reason, it soon became apparent that simply wiring the I/O pins of the FPGA directly to the pins of a programmable chip might not work properly and could quite possibly damage some programmable chips.

3.1 Buck Converter

To overcome this challenge a custom PCB is to be created with includes a buck converter to create a variable voltage supply to power different types of programmable chips. This portion of the custom PCB, the design of which is shown in Figure 2 above, will be connected to the Vcc pin of the programmable chip to supply the chip with power.

Initially, the buck converter was introduced into the custom PCB by using base components. This proved, however, to be a very difficult task and resulted in an initial design which did not perform particularly well in the LTSpice simulations. Thus, the design decision was made to implement a buck converter as shown in Figure 2 using the TPS560430Y buck converter chip by Texas Instruments, which can readily supply a variable output voltage (Vout) ranging from 1 Volt to 24 Volts. [4]

The input voltage depicted in Figure 2 is currently set at 12 V, though the TPS560430 buck converter chip is capable of operating with an input voltage from 4 Volts to 36 Volts. [4] Since buck converters work by stepping down the input voltage to a stable output voltage of a lesser value, this custom PCB could readily be hooked up to a higher input voltage in order to be capable of providing an output voltage of up to 24 Volts, which is moreso in the range which some Programmable Logic Controllers (PLC) require.

Texas Instruments provides an example testing module schematic for the TPS560430 family of buck converter chips. [5] This example schematic was greatly helpful in the effort to produce the buck converter portion of the custom PCB as depicted in Figure 2. One of the key differences between the Texas Instruments example testing module design and the custom PCB designed for this project is the inclusion of a digital potentiomer to control the output voltage of the buck converter circuit. This makes it possible to connect a few I/O pins of the FPGA to the digital potentiometer to set the output voltage of the buck converter digitally. The particular digital potentiomer chosen for this project is a Texas Instrument TPL0102-EP dual channel digital potentiomer with 256 resolution. [3] The wide resolution is particularly advantageous since it could therefore allow 256 different voltages levels from 1 Volt to 24 Volts with approximately 90 milli-Volts steps between each possible voltage level of Vout. The TPL0102-EP also stands out from alternative digital potentiometers in that it operates at a high frequency of 2.1 MHz. [3]

Another distinct advantage of the TPL0102-EP digital potentiometer is that it is dual channel. [3] Therefore, there are in fact two diggital potentiometers contained within one form factor. This opens up the possibility to later implement the ability to control the output voltage of a second buck converter to specify a very specific voltage for a second programmable chip. Such a feature could be useful for writing to multiple chips at once or to perhaps perform logic analysis on two chips simultaneously.

3.2 Current Mirror Considered

Throughout the design process of the custom PCB, a great deal of consideration was given to ensuring that the amount of current supplied to the pins of the programmable chip would fall within the safe current range as designated by each chip manufacturer. If too much current is supplied to the pins of the chip, then the chip might not be able to properly perform read or write functions. Too much current might permanently damage the chip.

To address these concerns, various methods for controlling current were investigated, the most promising of which was the NMOS current mirror depicted in Figure 3. Ultimately, this was not implemented into the design of the custom PCB, as it was later realized that this is unnecessary for the vast majority of the programmable chips. Even so, this work may prove useful for future iterations of this project if it later does become useful to precisely control current flow.

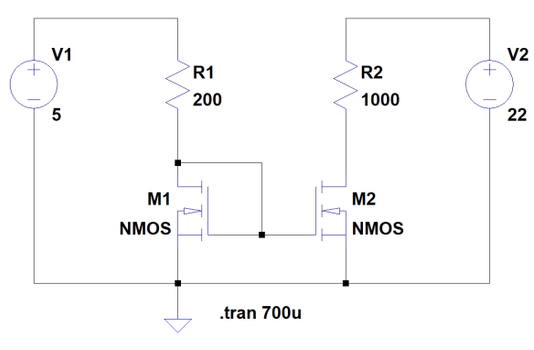


Figure 3: Current Mirror Considered

The NMOS current mirror circuit depicted in Figure 3 functions by allowing the designer to set the value of R1, in order to control the current which will flow through R2. Since the gate pins of both N-type MOSFETs are tied together, the current which flows through R2 will "mirror" the current which flows through R1. This could be quite useful in creating a circuit which has an unknown load that requires a precise current supply.

LTSpice was utilized to simulate the NMOS current mirror circuit. In the tests R2 was set at a constant value of 1 kΩ. The resistance of R1 was varied to simulate how the FPGA might control a digital potentiometer in place of R1 to vary the current, if the NMOS current mirror is later implemented into the design of the custom PCB. The results of these tests are summarized in Table 1.

R1 (kΩ)	I_{R1} (μA)	I_{R2} (μA)	V_{R2} (V)
0.2	245.12	245.12	0.25
1	227.74	227.74	0.23
5	171.57	171.57	0.17
10	133.97	133.97	0.14
100	32.09	32.09	0.03

Table 1: Current Mirror LTSpice Simulation

As can be seen in the LTSpice simulation data presented in Table 1, the current which flows through R2 does indeed match the current which flows through R1. It could be expected, however, that a replication of the experiment with physical components might yield slightly different results. Variance in the resistance of the R1 resistor could potentially be compensated for by calibrating the digital potentiometer which would be in place of R1. However, differences between the gain curves of the N-type MOSFETs could result in slightly different currents through R1 and R2, despite V_{GS} being the same for both of the MOSFETs.

Aside from the realization that the use of the NMOS current mirror within the custom PCB is unnecessary, there were concerns as to how the voltage across R2 would be affected by controlling the current through R2. After all, R2 represents the load of the chip. The initial plan was to provide a current mirror for each pin of the programmable chip so that each pin would be sure to have the precise amount of current recommended by the manufacturer. However, it became apparent that it would be difficult to control both the voltage and the current simultaneously and that it would be completely unnecessary to do so since the custom PCB would in effect be attempted to predict the load resistance of the programmable chip. The strategy thus became to ensure that each pin of the programmable chip is supplied with the precise amount of voltage recommended by the manufacturer, and then the resulting safe amount of current would inevitably flow through the chip, assuming that the custom PCB is provided with a power supply which can produce an adequate amount of current. This design decision greatly simplified the project and allowed further progress to be made.

3.3 Resolving Floating Pins

One of the interesting challenges faced when interacting with the programmable chips is the need to account for floating pins. Much like a radio antennae, a floating pin is not connected to a common ground of the overall circuit, neither is it being forced to necessarily represent what the overall circuit would consider to be a logic high, such as 5 Volts.

Thus, interacting with a chip which has floating pins can result in a great deal of ambiguity. For example, if the pins of the programmable chip are directly connected to the I/O pins of the FPGA, then what the chip is outputting as a logic low might not truly be what the FPGA considers to be ground. The voltage on the floating pins may rise to a voltage slightly above the common ground of the overall circuit. Similarly, when the programmable chip attempts to output a logic high, the voltage which the pin actually reaches may not at all match what the FPGA considers to be a logic high on its I/O pins.

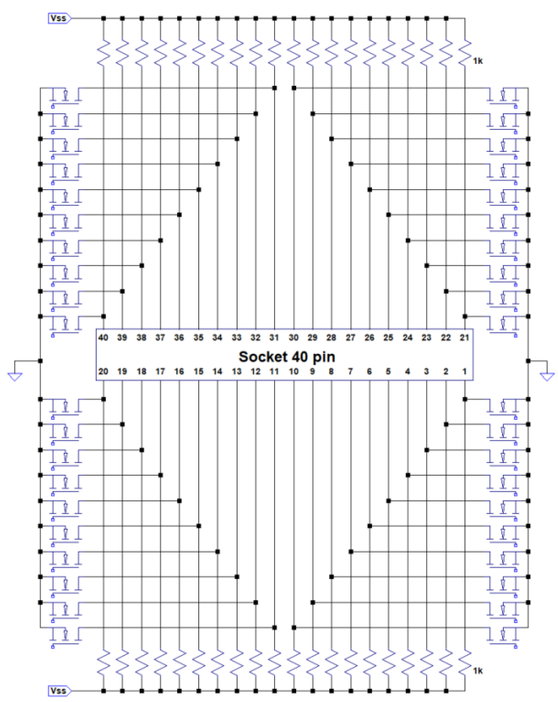


Figure 4: Pull-up Resistors and NMOS Gates

To resolve the issue of floating pins, pull-up resistors have been implemented into the design of the custom PCB, as depicted in Figure 4. Each pull-up resistor is connected to a voltage source (V_{ss}) which represents what the particular make and model of programmable chip expects as a logic high. For instance, if the data-sheet of a given programmable chip states that it considers 3.3 Volts to be a logic high, then V_{ss} would be set to 3.3 Volts.

In parallel with each pull-up resistor is an NMOS transistor. The gate pin of each NMOS is connected to a different I/O pin on the FPGA board. When an I/O pin on the FPGA is set to a logic high, then the NMOS transistor allows current to flow through the transistor, thus tying the pin to common ground. Conversely, when an I/O pin on the FPGA is set to logic low, then the gate pin on the NMOS is low and no current is permitted to flow through the NMOS transistor. Thus, when an I/O pin on the FPGA is low, the respective pin on the programmable chip is pulled up to logic high (V_{ss}) by the pull-up resistor. When the I/O pin on the FPGA is high, then the respective pin on the programmable chip is pulled down to common ground, logic low.

This configuration makes it possible to reliably perform write operations to the programmable chip. It should prove to be a simple, reliable solution to the challenge of floating pins.

4 Programmable Chip Selected

With so many different types of programmable chips on the market today, it became apparent that it may be useful to initially choose one specific model of programmable chip to focus upon. Electrically-Erasable Read-Only Programmable Memory (EEPROM) chips which communicate via the Inter-Integrated Circuit (I^2C) protocol are particularly common. Thus, the example chip chosen for this project is the M24128 family of 128 Kbit serial I^2C EEPROM chips by STMicroelectronics.

The datasheet for the M24128 family of chips reveal that the chip has 5 main modes of operation:

- write to identification page,
- lock identification page,
- read from identification page,
- read from memory array, and
- write to memory array. [7]

To better understand how the M24128 family of EEPROM chips function, a finite state machine was created to represent how the chip will step through each logical state. The resulting finite state machine diagram is depicted in Appendix A. Each of the 5 different possible operations for the chip have been highlighted in a unique color. The finite state machine provides a visual representation of the logic control structure for the M24128 family of EEPROM's, which could add a great deal of clarity when programming the FPGA to interact with this type of programmable chip.

5 Programming the FPGA

With an example chip chosen for the project, the M24128 family of serial I^{2}C EEPROM chips, the focus of the project shifted towards how the FPGA might be programmed. The FPGA would be responsible for reading from and writing to the programmable chips, utilizing the custom PCB to ensure that the correct voltages and currents are supplied to the chip.

Verilog or VHDL are the two main languages which can be used to program the FPGA. While Verilog code might be easier to produce, VHDL is more strongly-typed. The primary concern was that during development of this project if Verilog was used to program the FPGA then it might not throw an error and compile the code with a loose set of assumptions. VHDL, on the other-hand, is more strongly-typed and would likely not compile if a coding mistake was made. Thus, it was decided to use VHDL for the programming of the FPGA with the hopes that doing so would enable coding errors to be much more easily identified.

Initially, a great deal of time was spent attempting to learn VHDL and to implement the I^{2}C protocol so that the FPGA could communicate with programmable chips which utilize I^{2}C. This proved to be particularly challenging, especially when taking into account the timings of the rising edge of the internal clock of the FPGA in respect to the timing of the clock to be used for the I^{2}C communication.

Then it was realized that instead of implementing I^{2}C from scratch, a much more expedient option would be to use an open-source intellectual property (IP) core. After searching through various options, an open-source IP core which implements I^{2}C for Master devices was discovered through OpenCores. [6] The I^{2}C IP core is provided in VHDL format and takes into account the timings of the internal clock of the FPGA. While there is no documentation for the IP core and very few comments within the code, a better understanding of how it is meant to function was slowly gained through trial and error.

I^{2}C utilizes 2 wires to communicate between devices, referred to as SDA for the data line and SCL for the clock line. The particular I^{2}C IP core selected for this project is operated by a set of logical inputs and provides feedback through logical outputs. For this project, the FPGA is the Master device, since it is controlling what the programmable chip does. The programmable chip is therefore the Slave device. A visual summary of the logical and physical I/O of the I^{2}C IP core is depicted in Figure 5.

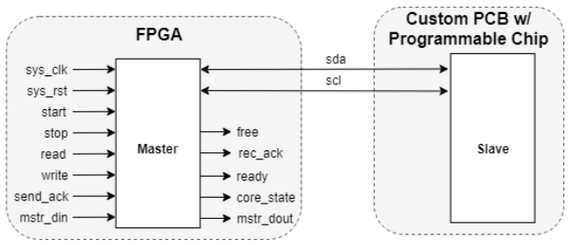


Figure 5: I^{2}C IP Core I/O Diagram

One exceedingly useful aspect of the I^{2}C communication protocol is that it can support multiple Slave devices. It could therefore be possible to easily connect multiple programmable chips to the FPGA to quickly write data to multiple chips at once. This could prove quite useful for manufacturers who may wish to mass produce a product which contains a programmable chip.

To ensure that a solid understanding of the open-source I^{2}C IP core has been gained, a finite state machine has been created to visually describe how the FPGA would utilize the I^{2}C IP core to interract with the programmable chip. The most recent iteration of this finite state machine is displayed in Appendix B. This provides a framework by which VHDL can be written to empower the FPGA to perform read and write operations with the M24128 family of serial I^{2}C EEPROM chips.

6 Future Plans

The project will be continued by members of the Security in Silicon Lab (SSL), the research lab which funded this project and whose members provided me with guidance throughout the development process. The custom PCB will be further developed and printed. Then initial testing can begin with the hardware components.

Adding in the ability for the FPGA to also serve as a logic analyzer could be quite useful. This could lessen the need for having on-hand an oscilloscope. The FPGA could also be programmed to perform many of the same functions performed by a Bus Pirate. Not only could adding such functions to the FPGA save money by reducing the need to buy additional equipment, but doing so could save time by negating the need to continuously move probe wires around.

Enabling the FPGA to read the entire memory contents of a programmable chip, then generate a hash of those contents could potentially be helpful for determining if a chip has been tampered with or if the

chip has the most recent version of the data. It may also be helpful to keep a digital record which lists the unique Identification Page number of each chip, followed by a list of all of the different hashes which have been generated from that chip's data. This could in essence provide a history of each time the chip's data was modified by the universal chip programmer.

Another possible area for future development of this project might be to include in the chip's memory array data or Identification Page a cryptographic key. This could provide some indication if the chip has been modified by a different programmable device, or by an unauthorized user.

7 Conclusion

Creating a low-cost universal chip programmer which supports a wide variety of different programmable chips could prove very useful for many people. Adding additional functionality to the FPGA, such as the ability to serve as a logic analyzer, could further add value to this project.

The LTSpice simulations and resulting circuit for the custom PCB have provided a foundation upon which this project may be continued further. Research into how the FPGA will be programmed to interact with various programmable chips has yielded finite state machines to describe both the functionality of the programmable chip itself and the I^2C IP core which will be running on the FPGA.

This project was exceedingly challenging, though a great deal has been learned. As the project continues to be developed and refined, perhaps it may enable a great many people to create their own universal chip programmers using relatively low-cost components.

8 Acknowledgements

I would like to thank the Security in Silicon Lab (SSL) for their continued guidance throughout the development of this project. The SSL has a presence in the University of Central Florida as well as the University of Florida. Dr. Yier Jin of the University of Florida is the faculty advisor for the Security in Silicon Lab.

I would also like to thank the Computing Research Association Women (CRA-W) for providing me with the opportunity to engage in this research project this summer. The CRA-W graciously provided me with financial support and connected me with a faculty mentor, Dr. Yier Jin, so that I could gain research experience prior to beginning my graduate studies. Thank you for believing in me!

References

[1] Terasic Inc. *DE10-Nano Kit*. URL: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=205&No=1046&PartNo=2#section. accessed: 07.26.2018.

[2] Xeltec Inc. *SuperPro 6100*. URL: https://www.xeltek.com/universal-programmers/superpro-6100-universal-ic-chip-device-programmer/. accessed: 07.25.2018.

[3] Texas Instruments. *256-Taps Dual-Channel Digital Potentiometer With Nonvolatile Memory*. URL: http://www.ti.com/product/TPL0102-EP. accessed: 07.26.2018.

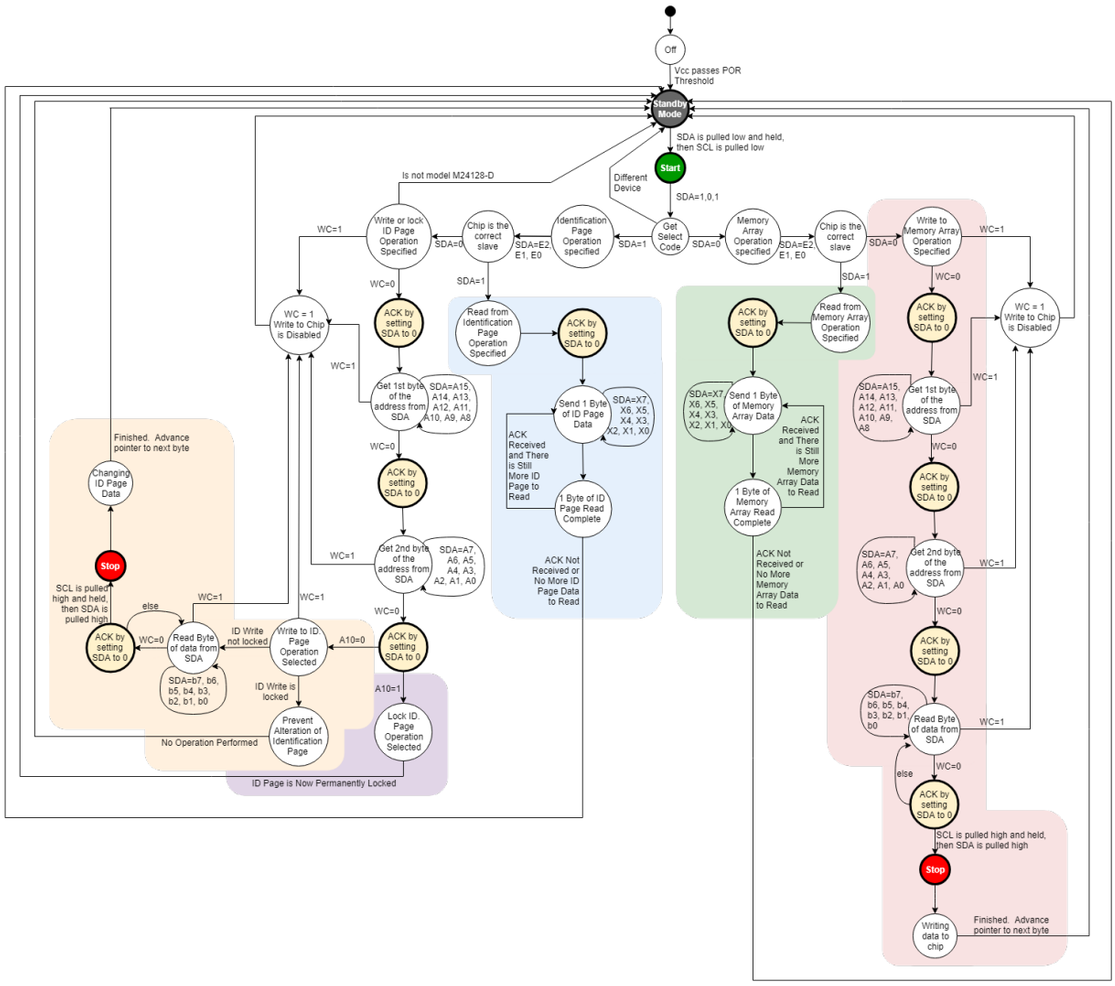
[4] Texas Instruments. *SIMPLE SWITCHER® 36-V, 600-mA Buck Regulator With High-Efficiency Sleep Mode*. URL: http://www.ti.com/product/TPS560430/description. accessed: 07.26.2018.

[5] Texas Instruments. *TPS560430 Evaluation Module*. URL: http://www.ti.com/lit/ug/slvub52/slvub52.pdf. accessed: 07.26.2018.

[6] Eli Smertenko. *I2C Master Slave Core*. URL: https://opencores.org/project/i2c_master_slave. accessed: 07.30.2018.

[7] STMicroelectronics. *128-Kbit serial I2C bus EEPROM*. URL: https://www.st.com/resource/en/datasheet/m24128-bf.pdf. accessed: 07.30.2018.

Appendix A Finite State Machine of M24128 Chip Operation From the Perspective of the Chip



Appendix B I^{2}C Master IP Core Finite State Machine