SCARAB: Research Document

For submission to SETU

Author: Stuart Rossiter

Student Number: C00284845

Course: BSc (Hons.) Software Development

Supervisor: Joseph Kehoe

Contents

1. Introduction

1.1 Abstract

The SCARAB device and accompanying program are retro game preservation tools designed to check the health of retro game cartridges, and manage their save data. The device, built around a microcontroller, interfaces with the cartridges via cartridge port modules. These modules can be swapped in and out, allowing for a variety of cartridge types, and providing future expansion options. The SCARAB device can auto-detect the inserted modules, and can detect whether a cartridge is inserted. The SCARAB program provides a GUI for the user to interact with. It provides options to dump and restore save data, run a full diagnostic checkup on an inserted cartridge, and run individual tests. It also serves as a save browser, allowing the user to view the save files they have dumped from cartridges.

1.2 Overview

When it comes to researching for the SCARAB, the research needed to be split up into 2 parts. One part focuses on the GUI PC Program. This first part goes into the choice of language for developing the program, and discusses the libraries which have been considered for use with the language. The second part of the research focuses on the SCARAB Device itself. First and foremost, research has been done about microcontrollers, as the “brain” of the device. In addition, the architecture of game cartridges will be discussed, as they will need to be interfaced with by the microcontroller. Finally, various electronic components will be required, and they will be discussed accordingly.

2. SCARAB PC Program

2.1 Languages

When it comes to choosing a language for the PC program portion of the SCARAB, a lot of thinking needed to be done. The list of requirements for a language were as follows:

- Does not need to be super performance oriented (e.g. Assembly).
- Needs to be able to interface with serial ports, to communicate with the SCARAB's Arduino.
- Needs to have a good GUI library.
- Needs to be cross platform (Windows, Linux, MacOS).

Several languages were considered for this.

Name	C++	C#	Python
Speed	Very High	High	Moderate
Serial Comm.	QSerialPort library	Native	pyserial library
GUI Library	Qt	MAUI / Avalonia	PyQt / PySide
Cross Platform?	Yes	Yes	Yes
Development Speed	Slowest	Slow	Fast

All of the above languages have their pros and cons. For this project, execution speed is not important, and development speed is always ideal.

2.2 Libraries

2.2.1 pyserial

The pyserial library, as the name suggests, is a library for Python which encapsulates the access for the serial port. Authored by Chris Liechti, pyserial has over 60 contributors, and 98k users on GitHub [1]. The chosen microcontroller (see 3.1) uses serial as its communication method with PC, so it necessitates the use of pyserial.

2.2.2 Qt/PySide

Qt is a cross-platform GUI toolkit, developed by the QT company. PySide, developed by the Qt Company itself, is a Python binding of the GUI toolkit "Qt". As Qt is cross-platform, applications built with PySide will run on any platform that supports both Qt and Python. This includes Windows, OS X, Linux, and even iOS and Android [2].

3. SCARAB Device

3.1 Microcontrollers

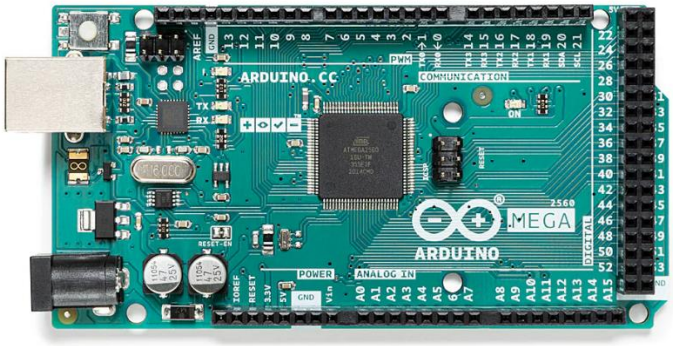
To serve as an interface between the cartridge port modules and the SCARAB PC program, some kind of microcontroller was deemed necessary. Microcontrollers are small computers on a single chip. They contain a processor core, RAM, and EEPROM, for storing programs to run. The purpose of a microcontroller is to manage a specific set of tasks within an embedded system, without the need for a complex operating system [3]. This is perfect for the SCARAB, as

there are only a handful of tasks necessary, and a dedicated machine or Raspberry Pi would be excessive. The brand of microcontroller I had settled on was Arduino.

3.1.1 Why Arduino?

Among the reasons for choosing Arduino, the main one was the variety of development boards. Several families of boards exist, such as Nano, MKR, UNO, Classic, and Mega, sporting over 30 different boards between them [4]. Given the sheer number of boards, and all their different configurations, there is sure to be a board suitable for the SCARAB. In addition to this, Arduino is inexpensive, has cross-platform support, and the software and hardware are completely open source. Choosing Arduino is not all that needs to be chosen, however. The most suitable board needs to be chosen, and in this case, it's the Arduino Mega 2560 Rev3.

3.1.2 Arduino Mega 2560 Rev3



(Fig 3.1.2.1 – Arduino Mega 2560 [5])

Based on the ATmega2560 microcontroller, the Arduino Mega 2560 Rev3 contains everything needed to support the ATMega2560, including 54 digital I/O pins, 16 analog inputs, 4 hardware serial ports, a 16MHz crystal oscillator, USB connection, and a power jack [6]. The Mega 2560 was the perfect choice for several reasons. Due to the sheer number of I/O pins required for some cartridges, only boards in the Mega family of Arduinos would be sufficient. Within the Mega family, the Arduino Mega 2560 Rev3 is the only one powered by 5V; the Due and GIGA use 3.3V. Seeing as most cartridges use 5V, this was the ideal choice.

3.2 Languages

For the Arduino, I see no point in using any language other than the default language used in the Arduino IDE. The Arduino Language is based on C++, with the addition of special methods and functions to interface with the board [7]. Given my experience in C++, and the native support by Arduino, there is no reason to switch to MicroPython or TinyGo.

3.3 Game Cartridges

The SCARAB is intended to interface with many different cartridges through its module boards. The architecture of these cartridges varies, so measures must be taken on a system-by-system basis.

3.3.1 Nintendo Entertainment System (NES)

Released in 1983 in Japan as the Famicom, and later in Europe in 1986 [8], the NES single-handedly saved the gaming industry. The cartridges, known as Game Paks (a recurring name for Nintendo), have since become iconic.

(Fig. 3.3.1.1 NES Cartridge External View)

(Fig. 3.3.1.2 NES Cartridge Internal View)

Earlier games, such as Super Mario Bros, were simple to read and write to, as they contained 2 important chips: one containing 32KB of Program ROM, and the other containing 8KB of Character ROM. These were part of a board revision known as NROM-256 [9]. The problem with reading later cartridges comes from the introduction of "Mapper Chips". These chips allowed the NES to bypass its 16-bit address bus limit, by swapping the currently accessible ROM data on the cartridge, allowing for much bigger games [10]. All but 20 games are covered by the Action 53, MMC1, and MMC3 families, so implementing the 256 registered mappers is not a goal of this project.

```
 (front/top)          (back/bottom)
      NES    | Cart  |    NES
             -------
      +5V -- |36   72| -- GND
  CIC toMB <- |35   71| <- CIC CLK
 CIC toPak -> |34   70| <- CIC +RST
    PPU D3 <> |33   69| <> PPU D4
    PPU D2 <> |32   68| <> PPU D5
    PPU D1 <> |31   67| <> PPU D6
    PPU D0 <> |30   66| <> PPU D7
    PPU A0 -> |29   65| <- PPU A13
    PPU A1 -> |28   64| <- PPU A12
    PPU A2 -> |27   63| <- PPU A10
    PPU A3 -> |26   62| <- PPU A11
    PPU A4 -> |25   61| <- PPU A9
    PPU A5 -> |24   60| <- PPU A8
    PPU A6 -> |23   59| <- PPU A7
 CIRAM A10 <- |22   58| <- PPU /A13
   PPU /RD -> |21   57| -> CIRAM /CE
      EXP 4   |20   56| <- PPU /WR
      EXP 3   |19   55|    EXP 5
      EXP 2   |18   54|    EXP 6
      EXP 1   |17   53|    EXP 7
      EXP 0   |16   52|    EXP 8
     /IRQ <-  |15   51|    EXP 9
   CPU R/W -> |14   50| <- /ROMSEL (/A15 + /M2)
    CPU A0 -> |13   49| <> CPU D0
    CPU A1 -> |12   48| <> CPU D1
    CPU A2 -> |11   47| <> CPU D2
    CPU A3 -> |10   46| <> CPU D3
    CPU A4 -> |09   45| <> CPU D4
    CPU A5 -> |08   44| <> CPU D5
    CPU A6 -> |07   43| <> CPU D6
    CPU A7 -> |06   42| <> CPU D7
    CPU A8 -> |05   41| <- CPU A14
    CPU A9 -> |04   40| <- CPU A13
   CPU A10 -> |03   39| <- CPU A12
   CPU A11 -> |02   38| <- M2
       GND -- |01   37| <- SYSTEM CLK
              -------
```

Figure 3.3.1.3 shows the pinout of the NES cartridge connector. Most of the important pins for this project come in the format "XXX YZ", where XXX is either CPU (leading to Program ROM), or PPU (leading to Character ROM), Y is either A for address line or D for data line, and Z is the number for the line. Other important pins include the 5V and GND lines, M2 for the mapper chips, /ROMSEL, CPU R/W, PPU /RD, and CIC +RST (Reset).

(Fig 3.3.1.3 NES cartridge pinout [11])

3.3.2 Super Nintendo Entertainment System (SNES)

In 1990, Nintendo Japan released the Super Famicom, the 16-bit successor to the Famicom. This was later released in Europe in June 1992 as the SNES [8].

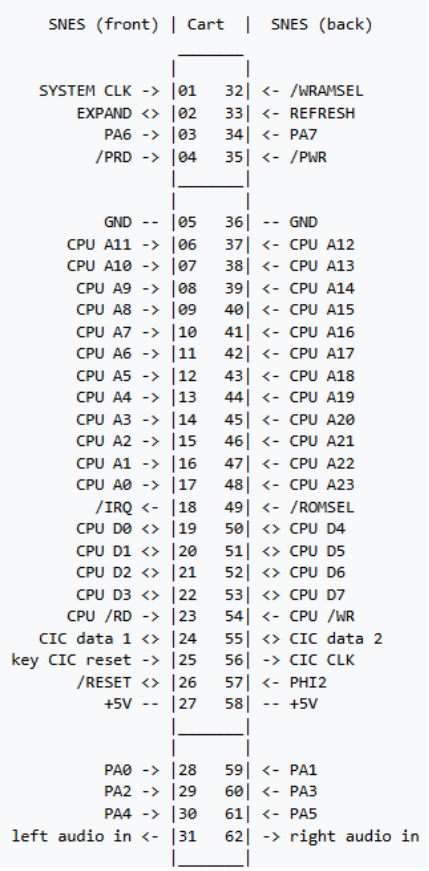
(Fig. 3.3.2.1 SNES Cartridges External View)

(Fig. 3.3.2.2 SNES Cartridges Internal View)

Many games are straightforward to read from and interface with. The problems arise with some of the "Enhancement Chips" that were seen in later games. Some enhancement chips hold no effect on the reading of cartridges, such as the Super FX chip (seen in Starwing, see MARIO CHIP-1 in Fig 3.3.2.2). The Super Accelerator 1 (SA1) chip, however, is troublesome.

The SA1 is a coprocessor, running at 10.74 MHz: 4 times faster than the speed of the SNES CPU [12]. As seen in figure 3.3.2.3, it has pins which interface directly with ROM, and has pins for CIC data. This is where the problem lies. Some ROM data is locked behind the SA1 coprocessor, as it contains a Super MMC memory mapper chip. The SA1 begins in a sleeping state, to be woken by the SNES CPU, which will not happen if the CIC lockout chip and correct clock timing are not made available to the SA1. This can be overcome, as seen in how the Sanni Open-Source Cartridge Reader works [13]. For the sake of this project, the SA1 bypass will be skipped. It may be implemented down the line.

(Fig. 3.3.2.3 SNES SA1 Chip Pinout [14])



(Fig 3.3.2.4 SNES Connector Pinout [15])

3.3.3 Nintendo 64 (N64)

1996 graced Japanese store shelves with the worlds first 64-bit home videogame system: the Nintendo 64. The following year, it was released in Europe, to critical acclaim [8].

(Fig. 3.3.3.1 N64 Cartridge External View)

(Fig. 3.3.3.2 N64 Cartridge Internal View)

The N64 marks the first cartridge type in my research to use 3.3V power and logic, as opposed to 5V. This is where bus transceivers and a buck converter will be needed see (3.4.1 and 1.4.2).

Cartridge

Looking at the top of the console with controller ports facing you. Pin 1 is lower left, pin 26 upper left.

Pin	Name	Description	Pin	Name	Description
1	GND		26	GND	
2	GND		27	GND	
3	AD15		28	AD0	
4	AD14		29	AD1	
5	AD13		30	AD2	
6	GND		31	GND	
7	AD12		32	AD3	
8	$\overline{WR}$	Write	33	ALE_L	Address latch low bits
9	3.3v		34	3.3v	
10	$\overline{RD}$	Read	35	ALE_H	Address latch high bits
11	AD11		36	AD4	
12	AD10		37	AD5	
13	12v		38	12v	
14	12v	expansion port	39	12v	Expansion port
15	AD9		40	AD6	
16	AD8		41	AD7	
17	3.3v		42	3.3v	
18	CIC_15	CIC → PIF Data	43	CIC_14	PIF <- CIC Data
19	CIC_11	1.95MHz clock (PIF driven)	44	?	JTAG_CLK_R4300 or $\overline{INT}$
20	$\overline{RESET}$		45	$\overline{NMI}$	
21	EEPROM_DAT		46	$\overline{CSYNC}$	
22	GND		47	GND	
23	GND		48	GND	
24	SL	Audio Left	49	SR	Audio Right
25	GND		50	GND	

(Fig. 3.3.3.3 N64 Connector Pinout [16])

Figure 3.3.3.3 shows the pinout of the N64 cartridge. AD0-15 are multifunctional. They control the upper 2 bytes of the address input, the lower 2 bytes of the address input, and the 2 bytes of data output. The addresses are latched within the cartridge by the ALE_L and ALE_H pins, and the data is read with the RD pin.

In addition to those, the GND, 3.3V, RESET, and EEPROM_DAT lines will likely be required.

3.3.4 Game Boy (GB)

In 1989, Nintendo Japan released the first handheld game system with interchangeable game cartridges. Its name? The Game Boy. It released alongside Tetris, one of its bestselling games. In 1990, the Game Boy came to Europe, and with it, Nintendo of Europe was formed [8].

(Fig. 3.3.4.1 Game Boy Cartridge External View)

(Fig. 3.3.4.2 Game Boy Cartridge Internal View)

The Game Boy, much like the NES, had mapper chips of its own. Games larger than 32KB needed to use these mapper chips. These chips allowed for “banks” of ROM data to be switched in and out, for access via the address lines. This could be achieved by writing specific values to areas on the cartridge [17]. Apart from that, addressing the ROM is rather straightforward.

Cartridge

Pins numbered left to right, cartridge label up.

Pin	Function	Pin	Function	Pin	Function	Pin	Function
01	+5V	11	Address Bit 5	21	Address Bit 15	31	AUDIO IN
02	CLOCK	12	Address Bit 6	22	Data Bit 0	32	GROUND
03	$\overline{\text{WRITE}}$	13	Address Bit 7	23	Data Bit 1		
04	$\overline{\text{READ}}$	14	Address Bit 8	24	Data Bit 2		
05	$\overline{\text{CHIP SELECT}}$	15	Address Bit 9	25	Data Bit 3		
06	Address Bit 0	16	Address Bit 10	26	Data Bit 4		
07	Address Bit 1	17	Address Bit 11	27	Data Bit 5		
08	Address Bit 2	18	Address Bit 12	28	Data Bit 6		
09	Address Bit 3	19	Address Bit 13	29	Data Bit 7		
10	Address Bit 4	20	Address Bit 14	30	$\overline{\text{RESET}}$		

(Fig. 3.3.4.3 GB Connector Pinout [18])

Above is the pinout for the Game Boy cartridge. All but AUDIO IN will likely be required.

3.3.5 Game Boy Color (GBC)

1998 saw the release of the next step up from the Game Boy, in the Game Boy Color [8].

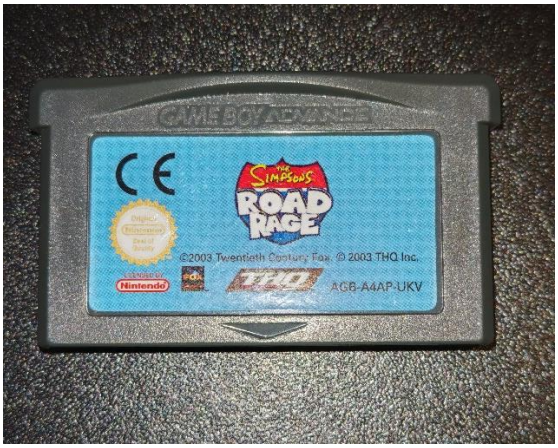
(Fig. 3.3.5.1 GBC Cartridge External View)

(Fig. 3.3.5.2 GBC Cartridge Internal View)

From my research, the Game Boy Color cartridges utilise the same addressing and banking systems that the original Game Boy used. This makes things easier, as they are essentially Game Boy games, but bigger.

3.3.6 Game Boy Advance (GBA)

The Game Boy Advance released worldwide in 2001, being marked as the fastest selling console ever with over 500k units sold in its first week in Europe. 2 years later, we would see it take on a new clam-shell form; the Game Boy Advance SP [8].



(Fig. 3.3.6.1 GBA Cartridge External View)

(Fig. 3.3.6.2 GBA Cartridge Internal View)

GBA cartridges are the second cartridges to use 3.3V power and logic. So long as the first read is non-sequential, it's possible to sequentially read from the ROM by pulsing the "RD" pin repeatedly, to a point [19].

Pin	Function	Pin	Function	Pin	Function	Pin	Function
01	+3.3V	11	Data/Address Bit 5	21	Data/Address Bit 15	31	IRQ
02	CLOCK	12	Data/Address Bit 6	22	Address Bit 16	32	GROUND
03	$\overline{\text{WRITE}}$	13	Data/Address Bit 7	23	Address Bit 17		
04	$\overline{\text{READ}}$	14	Data/Address Bit 8	24	Address Bit 18		
05	$\overline{\text{CHIP SELECT}}$	15	Data/Address Bit 9	25	Address Bit 19		
06	Data/Address Bit 0	16	Data/Address Bit 10	26	Address Bit 20		
07	Data/Address Bit 1	17	Data/Address Bit 11	27	Address Bit 21		
08	Data/Address Bit 2	18	Data/Address Bit 12	28	Address Bit 22		
09	Data/Address Bit 3	19	Data/Address Bit 13	29	Address Bit 23		
10	Data/Address Bit 4	20	Data/Address Bit 14	30	$\overline{\text{CHIP SELECT 2}}$		

(Fig. 3.3. GBA Connector Pinout [20])

Reading the ROM is simple enough. The save data depends on the format, however. If it is battery backed SRAM, it's simple reads and writes. Flash storage requires specific "commands" sent as writes to memory [21]. EEPROM is easily the most annoying of the 3. To make a long story short, it handles data serially, i.e. one bit at a time [22].

3.4 Electronic Components

3.4.1 Bus Transceiver

As the GBA and N64 cartridges utilise 3.3V logic, as opposed to the Arduino's 5V logic, some form of voltage lowering was required. The method of choice was the SN74LVC245AN Octal Bus Transceiver. The bus transceiver allows for bi-directional voltage modulation, converting 5V to 3.3V on one side, and 3.3V to 5V on the other. This feature lets the device function as a translator in mixed 3.3V and 5V environments [23]. These transceivers will allow the Arduino's 5V logic to interface with the 3.3V logic of the GBA and N64 cartridges.

3.4.2 Buck Converter

Not only do the GBA and N64 cartridges use 3.3V logic, but they utilise 3.3V VCC too. The transceivers would not work for the level of amps required, so a new solution was needed. This

is where buck converters are introduced. Buck converters convert a DC voltage to a lower DC voltage, such as 5V to 3.3V in this case [24].

3.4.3 USB-C Input Module

While the Arduino Mega 2560 has enough pins to interface with the cartridge, it doesn't support the Amps necessary to also power the cartridge. Using a USB-C cable, the SCARAB could draw the power necessary from a USB port on the PC. While this will require 2 USB cables to be connected between the SCARAB and the PC, it's preferable to the alternative of batteries.

3.4.4 Cartridge Ports

The Arduino has many I/O pins, but cartridges can't be inserted directly into pins. Cartridge ports are widely available, mostly known by their pin count. For example, the NES port is known as a 72-pin connector. These ports have pins which can be slotted into pin connectors, or directly into PCBs, which will be the case for the cartridge port modules.

3.4.5 Resistors

Resistors are passive devices used to control the flow of current in a circuit. In addition to this, they can divide the voltage of a circuit [25]. Both are useful for this project. Controlling current flow helps to prevent damage to fragile components. The voltage division, however, will allow for the detection of the currently inserted cartridge port module. By using different resistor combinations on each module, the output of the voltage divider can be measured against a table of existing modules to determine the currently inserted one.

4. Similar Products

4.1 Open Source Cartridge Reader (OSCR)

Created by Sanni, the Open Source Cartridge Reader is a device designed to allow the dumping of cartridge ROMs and Save Data, along with the restoring of saves.

Similarities:

- Both the SCARAB and OSCR will dump and restore save data.
- Both the SCARAB and OSCR use Arduinos from the Mega family.
- Both the SCARAB and OSCR can calculate the checksum of a cartridge.

Differences:

- The OSCR does not check the checksum against a known valid checksum.
- The OSCR does not interface with a PC. It's a standalone device with an LCD and SD card. The SCARAB has a PC program it interacts with.
- The SCARAB has automatic voltage detection as standard, with the OSCR it is a modification.
- The OSCR focuses on software preservation, and the SCARAB focuses more on hardware preservation.
- The OSCR solders in multiple cartridge ports to the device itself, where the SCARAB uses a plug-and-play module system.

4.2 Retrode2

The Retrode2 is a USB interface for retro cartridges and controllers. It allows users to play their favourite retro games on PC emulators.

Similarities:

- The Retrode2 allows for save dumping and restoring.
- The Retrode2 interfaces with a PC.

Differences:

- The Retrode2 does not check the integrity of the cartridges.
- The Retrode2 serves as an adaptor for SNES and Sega controllers to PC.
- The Retrode2's main purpose is to bridge the gap between the cartridges and emulators on modern machines, where the SCARAB's main purpose is hardware preservation.

5. Conclusion

Throughout this document, many aspects of the SCARAB have been researched and discussed. First was the GUI Program, where the candidates for language were discussed. Python was chosen as the language for the program, with the pyserial and PySide libraries being required. As far as the SCARAB device is concerned, all aspects of it have been researched. The microcontroller I am choosing is the Arduino Mega 2560. The language of choice for the Arduino is the default C++ based Arduino language. Space conservation is important, so MicroPython won't do, and TinyGo is focused on concurrency, but the Mega 2560 only has one core. Cartridge pinouts and architecture were also discussed. This was a very important topic, as the Arduino must interface with the cartridges. Knowing which pins are necessary, and where they lie on the connectors, along with any other oddities relating to the cartridges is paramount. Extra components are required, as research had found. These were simple things, such as bus transceivers and buck converters to handle voltage differences, cartridge ports to provide a more familiar interface for the Arduino, USB-C inputs for power, and resistors for many different things.

This document has established several ideas which are important to remember for the development of this project:

- Cartridges are complex. Voltages are different. Pins are different. Pin count is different. Some have extra chips such as mapper chips, EEPROMs, and even CPUs.
- Sometimes, simplicity is ok. For example, C++ may have better speed and memory control than Python, but none of that is needed for this project.

6. Appendix

7. Glossary

SCARAB – Save and Cartridge Aid Requiring Adapter Boards

GUI – Graphical User Interface

RAM – Random Access Memory

EEPROM – Electrically Erasable Programmable Read-Only Memory

I/O – Input/Output

ROM – Read Only Memory

CPU – Central Processing Unit

PPU – Pixel Processing Unit

SRAM – Static Random Access Memory

OSCR – Open Source Cartridge Reader

LCD – Liquid Crystal Display

SD – Secure Digital

USB – Universal Serial Bus

DC – Direct Current

CIC – Checking Integrated Circuit

8. Bibliography

[1] GitHub. (2022). *pySerial*. [online] Available at: https://github.com/pyserial/pyserial [Accessed 12 Oct. 2025].

[2] Python GUIs. (2025). *PySide*. [online] Available at: https://www.pythonguis.com/topics/pyside [Accessed 12 Oct. 2025].

[3] Schneider, J. and Smalley, I. (2024). *What is a microcontroller? | IBM*. [online] www.ibm.com. Available at: https://www.ibm.com/think/topics/microcontroller [Accessed 10 Sep. 2025].

[4] Arduino (2022). *Arduino Hardware*. [online] www.arduino.cc. Available at: https://www.arduino.cc/en/hardware [Accessed 9 Oct. 2025].

[5] Arduino (n.d.). *Arduino Mega 2560*. Available at: https://store-usa.arduino.cc/cdn/shop/files/A000067_00.front_1000x750.jpg?v=1727102662 [Accessed 11 Oct. 2025].

[6] Arduino® MEGA 2560 Rev3. (n.d.). [online] Arduino. Available at: https://docs.arduino.cc/resources/datasheets/A000067-datasheet.pdf [Accessed 10 Oct. 2025].

[7] Kumar, A. (2023). *Why do We Use the Arduino Programming Language? How is it Helpful?* [online] Emeritus Online Courses. Available at: https://emeritus.org/blog/coding-arduino-programming-language [Accessed 12 Oct. 2025].

[8] Nintendo (2016). *Nintendo History*. [online] Nintendo of Europe AG. Available at: https://www.nintendo.com/en-gb/Hardware/Nintendo-History/Nintendo-History-625945.html [Accessed 11 Oct. 2025].

[9] NESdev Wiki. (2024). *NROM*. [online] Available at: https://www.nesdev.org/wiki/NROM [Accessed 11 Oct. 2025].

[10] NESdev Wiki. (2023). *Mapper*. [online] Available at: https://www.nesdev.org/wiki/Mapper [Accessed 11 Oct. 2025].

[11] NESdev Wiki. (2023). *Cartridge connector*. [online] Available at: https://www.nesdev.org/wiki/Cartridge_connector [Accessed 11 Oct. 2025].

[12] SnesLab. (2025). *SA-1*. [online] Available at: https://sneslab.net/wiki/SA-1#Super_MMC [Accessed 12 Oct. 2025].

[13] sanni (2024). *Reading SNES SFC carts*. [online] GitHub. Available at: https://github.com/sanni/cartreader/wiki/Reading-SNES-SFC-carts [Accessed 12 Oct. 2025].

[14] SNESdev Wiki. (2025). *SA-1 Pinout*. [online] Available at: https://snes.nesdev.org/wiki/SA-1_Pinout [Accessed 12 Oct. 2025].

[15] SNESdev Wiki. (2022). *Cartridge connector*. [online] Available at: https://snes.nesdev.org/wiki/Cartridge_connector [Accessed 12 Oct. 2025].

[16] ConsoleMods Wiki. (2022). *N64 Connector Pinouts*. [online] Available at: https://consolemods.org/wiki/N64:Connector_Pinouts [Accessed 12 Oct. 2025].

[17] Retrocomputing Stack Exchange. (2019). *How does the Gameboy's memory bank switching work?* [online] Available at: https://retrocomputing.stackexchange.com/questions/11732/how-does-the-gameboys-memory-bank-switching-work [Accessed 12 Oct. 2025].

[18] ConsoleMods Wiki. (2022). *Game Boy Connector Pinouts*. [online] Available at: https://consolemods.org/wiki/Game_Boy:Connector_Pinouts [Accessed 12 Oct. 2025].

[19] Ziegler, R. (2025). *Gameboy Advance*. [online] Mirrors.gg8.se. Available at: https://reinerziegler.de.mirrors.gg8.se/GBA/gba.htm [Accessed 12 Oct. 2025].

[20] ConsoleMods Wiki. (2022c). *GBA Connector Pinouts*. [online] Available at: https://consolemods.org/wiki/GBA:Connector_Pinouts [Accessed 12 Oct. 2025].

[21] Korth, M. (2024). *GBATEK - GBA/NDS Technical Info*. [online] Problemkaputt.de. Available at: https://problemkaputt.de/gbatek.htm#gbacartbackupflashrom [Accessed 12 Oct. 2025].

[22] DenSinH (2021). *[GBA] EEPROM Save Type*. [online] Dennis H. Available at: https://densinh.github.io/DenSinH/emulation/2021/02/01/gba-eeprom.html [Accessed 12 Oct. 2025].

[23] Texas Instruments (2015). *SN74LVC245A Octal Bus Transceiver With 3-State Outputs*. [online] *Texas Instruments*. Available at: www.ti.com/lit/ds/symlink/sn74lvc245a.pdf [Accessed 11 Oct. 2025].

[24] Yates, J. (2024). *Understanding Buck and Boost Converters and the Capacitors Behind Them*. [online] blog.knowlescapacitors.com. Available at: https://blog.knowlescapacitors.com/blog/understanding-buck-and-boost-converters-and-the-capacitors-behind-them [Accessed 11 Oct. 2025].

[25] Dahl, Ø.N. (2023). *What Is A Resistor And What Does It Do?* [online] Build Electronic Circuits. Available at: https://www.build-electronic-circuits.com/what-is-a-resistor [Accessed 11 Oct. 2025].