Gameboy flash cartridge

Contents

[Foreword 3](#_Toc48253021)

[Description 4](#_Toc48253022)

[Requirements 5](#_Toc48253023)

[User requirements 5](#_Toc48253024)

[Functional requirements 6](#_Toc48253025)

[Non-functional requirements 7](#_Toc48253026)

[General idea 8](#_Toc48253027)

[Cartridge hardware 8](#_Toc48253028)

[Cartridge firmware 8](#_Toc48253029)

[GB User Interface 9](#_Toc48253030)

[Cartridge Iterations 10](#_Toc48253031)

[Model PT - Prototype Test 10](#_Toc48253032)

[Hardware 10](#_Toc48253033)

[Fixes for next iteration 14](#_Toc48253034)

[Model PM - Prototype Modular 16](#_Toc48253035)

[Hardware 16](#_Toc48253036)

# Foreword

This document is a general overlook on the project. Here I explain mostly how my thought process went and which steps I took to get to where I am now. Additional references are provided to those who are interested in helping/making something cool which my sources (I’d love to see someone taking my work and expanding on it).

For a bit of context: I’m a HBO student (sort of University, but not quite) in the Netherlands studying ICT & Technology. During my study, I learned to program (C#, C and C++) and also learned how to work with microcontrollers and the like (among other things). I love working with low level stuff and am enthusiastic about learning about all kinds of different technologies.

During the Corona pandemic, I was looking for a way to fill my time. A year prior, I stumbled on the documentation for the GB. As I’m quite curious about older hardware and wanted to study more about the inner workings of computers/processors, I was quite interested in developing something for the GB. I didn’t really want to make a game (I’m not a game developer) and I didn’t have any other idea (or time) to use this information, so I quickly canned it.

After I bought a used GBC on ‘Marktplaats’ (basically eBay in the Netherlands), I was looking for ways to use it. I wanted to experience the hardware first-hand by playing some games on it and upgrade the thing to be a bit more modern (Li-ion battery mod and IPS screen mod), I saw a flashcard for it. As I was looking at it, I started to think about how that darn thing works, which led me to the path of trying to make one. I also had quite some free time during the pandemic, so that helped seal the deal.

# Description

The project goal, is to develop a flash cartridge for GB/GBC like the one in the picture below. The flash cartridge should be able to load games from a library on a common storage medium (best candidate is SD card) to play on the handheld. I want the cart to be compatible with GB and GBC (I may think about the GBS, but that is currently not the focus).

The project is separated into 3 parts: the cart hardware, the cart firmware and a selection menu software. I will need a game cart, capable of accommodating an SD card and additionally needed hardware (RTC and possibly more). I will also need some firmware to load the game, communicate with the SD card and probably some other stuff. And I will need a way for a user to select a game from the flash card for the cart to load.

  
A picture of the EZ-FLASH GB flash card. Source: <https://pt.aliexpress.com/item/4000718442858.html?gclid=CjwKCAjwsan5BRAOEiwALzomXyJAFjVWtUV81ep-307vMERlUKuTuoNc2pClN74Jw2VceHXs3LoyCxoCID4QAvD_BwE>

I currently have no real experience in developing hardware, so I will probably make a lot of mistakes in the electrical part (I’m a Software Engineer in training). My plan is to research the GB hardware to get an idea of what I need, then make a prototype cart, in order to test some firmware on it and get a better idea of what I need. Then, I’ll basically repeat this procedure, till I get the result I want.

# Requirements

As stated in [Description](#_Description), the ultimate goal is to create a cart that can load games from an SD card. For development, I need to break the project down into smaller parts.

## User requirements

1. **As a user, I must be able to select a GB game file on the SD card, in order for me to play it.**This is the basic requirement I need the cart to do. When a user insert the cart and boots the handheld, it should boot to a game selection menu of sorts, where a user can select his/her game.  
     
   In order to do this, I will need to be able to communicate with the cart to firstly get the game list, then by the user’s selection, load said game into some sort of buffer, as the GB can’t read directly from the SD card (refer to Research document ‘Bus decoding’ as to why). While the cart is responsible for loading games, I will also need a user interface to run on the GB. This will be the first thing that the GB loads when booting from the cart.
2. **As a developer, I want to be able to write my own firmware versions, in order to add functionality to the cart**As I’m a curious soul and want to spread that to the masses, I want to share the knowledge acquired from the project and document everything important for other people to develop their own firmware version (or maybe their own cart).  
     
   I will share documentation and other files regarding the development of this cart and publish it somewhere on the internet.
3. **As a user, I want to be able to relatively easy change the look of the user interface, in order to customise my experience**I like to customise my menu’s (be it my desktop, phone or 3DS). I want to add the functionality to change the look of the game selection menu (fonts, splash screen, BGM, you name it).  
     
   It’s a lower priority, but I will try to make a desktop program to make a ‘theme’ file, which you can put on the SD card to change the look of the UI.
4. **As a user, I want to be able to create backups of my save files, in order not to lose progress.**As someone who enjoys games, I despise it when I lose my saves. Now it’s becoming harder to lose said saves, with cloud storage and all (thank god), so I want to take that modern approach and use it on my platform.  
     
   It will probably take quite some work and it’s not really important, but it would be nice to have some kind of service that the cart can connect to, in order to backup saves (some kind of OneDrive). It would be cool if the cart has Wi-Fi hardware, so that it could communicate with such a service.
5. **As a user, I want to be able to play multiplayer games that I would normally play using the Link interface or IR interface, but then over Wi-Fi, in order to play with people all over the world.**Quite ambitious, but it is a cool idea. If I could figure out how to hijack the Link or IR interfaces, I could reroute the data from those interfaces through a network socket. Theoretically, one could then connect with users al over the globe and play multiplayer.  
     
   As this a HUGE requirement, it will receive a low priority. In order to do this, I would probably make some kind of service that clients can use to connect to one another. Same as the backup requirement, it takes a lot of time and effort to create such a think and it’s something to do, when the cart itself doesn’t have anything else going on.

## Functional requirements

Based on the user requirements and some research, I formulated these requirements. These requirements describe the more detailed requirements on a system level, needed to fulfil the user requirements.

1. **The GB and cart must be able to communicate with each other**To instigate certain behaviours in and to pass data to both the GB and cart (think about the cart telling the GB which games are available or the GB telling the cart which game it has selected), the GB and cart will need to be able to communicate with each other.   
     
   As the interface in between the GB and cart is the address and data lines, the cart will need to use a bit of the address space as a communication medium. I will probably use a bit of the cart RAM, but I will think about it when the time comes.
2. **The cart needs to communicate with the SD card**To read games and save data to non-volatile storage, the cart will need to communicate with the SD card.  
     
   Communicating with an SD card goes over an SPI connection. The cart will then need SPI functionality. For the software part, I will probably use some open source library. Writing it myself seems unnecessary. Although, it might be a fun learning experience.
3. **The cart will need some ‘brains’ to execute certain commands**To do stuff like communicate with the SD card and GB, read and write data, the cart is going to need some central processing. Also for me to customise the behaviour of the cart and because it needs to do some complex stuff, it seems wise to do this software based.  
     
   I will probably slap a microcontroller on the cart to do these things. Additionally, the research till now, suggests I will need some hardware acceleration, so an FPGA might also be handy. I’ve never used one before, but I’ve been wanting to learn to develop for an FPGA, so again, an excuse to do something new! I currently looking into something called PSoC.
4. **The cart will need a memory buffer to act as cart ROM and RAM**The GB uses a parallel 16-bit address and 8-bit data bus to communicate with what normally is the ROM (and RAM) on the cart. As the speed with which the GB reads from and writes to the cart is quite fast, it would be wise to have a buffer in between the data on the SD card and the GB. This would ensure both speed and minimise wear on the flash medium.  
     
   I will probably use some RAM chips to act as the memory buffer. It’s fast, reliable and resistant to the wear and tear from every game load.
5. **The cart will need an RTC unit**Some games on the GB use an RTC. This is a piece of hardware, which can keep track of the date and time. This is also a convenient thing to have, for the purpose of displaying time.  
     
   The easiest thing to have, is a MCU that has an integrated RTC. This is a common thing, so it shouldn’t be hard to find one.

## Non-functional requirements

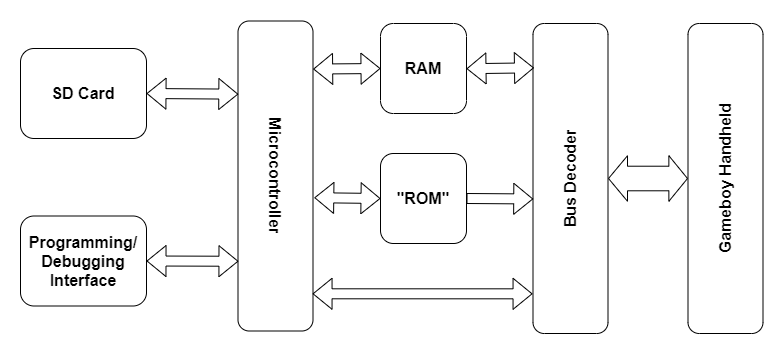
1. **Cart should use least power possible**Because the GB is known for being low power, I want my cartridge to use the least amount of power possible. Use of the cartridge shouldn’t drain 2 AA batteries in an abnormal way.  
     
   This should be achieved by carefully writing the cart firmware and disabling any peripherals not needed at the moment. I could also choose low power chips inside the cart.

# General idea

Here I describe the general idea of how I want to design my system. This includes all parts of my system and how they communicate/interact with each other. This will be quite rough, but should serve as a strong base to build upon.

## Cartridge hardware

The cart can be separated into 3 main parts: the Bus decoder, the memory and the MCU (Microcontroller). See below for an image visualising the connections between the parts.

  
Diagram showing the different parts of the cartridge and how they can interact.

The Bus decoder is the part which communicates with the GB. Its job is to decode the read/write requests from the GB and execute them. For a more detailed explanation about the decoder, see **TODO: Write detailed explanation about decoder**

The memory is where the game code and data reside. This is made out of 2 parts: the “ROM”, which is the part of the RAM that acts as ROM; and the RAM, which is also RAM and can be used by the GB as cart RAM. For a more detailed explanation about the memory, see **TODO: Write detailed explanation about memory**

And finally, the MCU. This part runs the firmware code and is responsible for the more complex behaviour of the cart. This part also communicates with the SD card, for loading in game code and data. For a more detailed explanation about the MCU, see **TODO: Write detailed explanation about MCU**

## Cartridge firmware

The firmware is the code running on the MCU which resides on the cart. To facilitate the execution of certain behaviours, it seemed wise to have a MCU with updateable firmware. The original plan is to have a MCU with a bootloader, which reads a firmware file on the SD card and executes this firmware.

The general stuff the firmware does is as follows:

* When starting up, read and download the firmware file on the SD card (done by the bootloader)
* Upload the menu program for the GB into the cart memory
* Configure the decoder for use with menu program
* Enter in a mode, which waits for requests from the GB (e.g. load selected game, get or set the date/time)
* When a game has been selected, upload selected game (and optionally save data) into the cart memory
* Configure decoder for use with selected game
* Enter power saving mode, if MCU is not needed anymore
* When GB powers off, save the RAM to a file on the SD card

For a more detailed explanation about how the firmware works, refer to the code. I’ve written enough comments to adequality explain the inner workings.

**TODO: Write the damn code**

## GB User Interface

# Cartridge Iterations

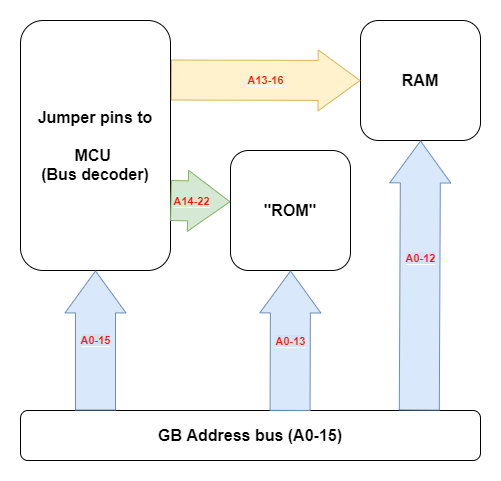
This chapter is to show the different iterations I’ve made of the cartridge and to discuss how they works, why I did certain things and what I would change about them in the next iteration.

## Model PT - Prototype Test

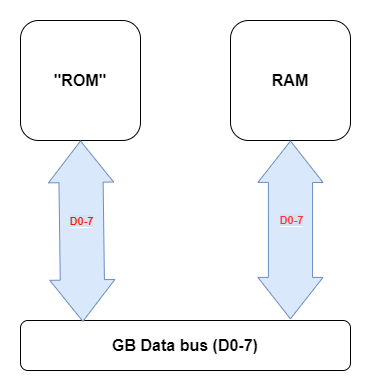
This is the first prototype made for this project. As I’ve wanted to experiment with the memory, I’ve let out the MCU out of this design. I don’t have any experience in designing a board for a MCU and I didn’t want to have complications because of that. Also, I haven’t decided on the MCU that will run the show and I wanted to do this after I have a better picture about using the memory. This iteration also doesn’t have an SD card reader for the same reason. MCU and SD card reader should be added externally.

### Hardware

This design contains 2 RAM chips: a 64Mbit and a 1Mbit chip. The 64Mbit is to be used as the “ROM” section of memory and the 1Mbit chip as the RAM section. Both chips are connected to the address bus coming from the GB to access their part of the memory map (see research document for an explanation why **TODO: Write said explanation**). Because the chips are much bigger then their allocated part in the memory map, they have a couple extra address lines to address the banked memory. The extra lines are connected to jumper pins, which can be connected to a MCU for the higher addresses (See picture below for diagram showing the address pins wiring). When the GB then wants to access different banks, the higher bits should then be set by the MCU, in order to access these banks.

  
Image showing wiring of address lines.

Additionally, the data bus of both chips are also connected to each other and to the GB data bus (See image below for wiring diagram). As these lines can be driven by the RAM chips, care should be taken that they don’t try to use the data bus at the same time. This can be achieved by using the ‘Output Enable’ and ‘Chip Enable/Select’ pins of each chip (accessible via the jumper pins). Still, read their datasheets for a more detailed explanation of their inner workings.

  
Image showing wiring for data bus.

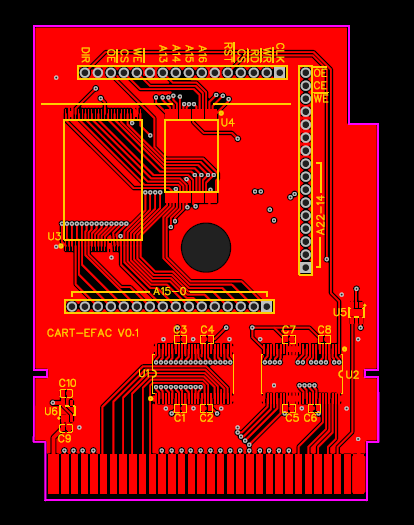
I couldn’t find RAM chips of adequate size that work on 5V, So I’ve had to settle on chips that work on 3.3V. Because of this, I needed some components as voltage translators. The most notable are the bus transceivers, which are mainly used as down translators that translate the 5V logic signals coming from the GB to 3.3V signals (address bus, data bus and control signals). They have another function, which is to isolate the GB when the MCU wants to access the RAM chips. During some operations (e.g. when loading in the selected game) the GB shouldn’t be able to access the cart memory, so the bus transceiver can be used to block the GB’s signals. This can be achieved manipulating the control pins of the bus transceivers (see their respected datasheet for how they operate).

Because of the fact the RAM works with 3.3V logic, I’ve made the decision to use only 3.3V logic on the cartridge. This should simplify communication between components on the chip (I don’t have to constantly translate voltages). The only isolation still needed by the cart, is the RESET line. This is a line the cart can pull down to reset the GB. As this is also 5V logic, I’ve isolated it from the rest of the cart, using a level shifter. The level shifter can be used to pull the RESET line down.

To use the cart, I’ve exposed some connections by using jumper pins. These connect to nets, which I’ve seen as useful to have access to. These pins should then be connected to a microcontroller. **WARNING: All pins connect to 3.3V logic. You should NOT put 5V on these lines.** Doing so will most probably damage components on the board. There are 3 sets of 8 pins. The unlabelled pins are connected to ground. For description on the labelled pins, please refer to the next table.

|  |  |  |
| --- | --- | --- |
| **Pin(s)** | **Name** | **Description** |
| DIR | Direction | Change the input/output direction of data transceiver. |
| OE | Output Enable | Enable the output of data on the 1Mbit RAM chip. |
| CS | Chip Select | Enable 1Mbit RAM chip. |
| WE | Write Enable | Specify a read or write operation on the 1Mbit RAM chip. |
| A13-16 | Address lines | Higher address lines of the 1Mbit RAM chip. |
| RST | Reset | GB reset line (connected through level shifter). Pull down to reset GB. |
| CS | Chip Select | GB chip select line (down-translated to 3.3V). Active LOW. When active, GB is communicating with cart. |
| RD | Read | GB read line (down-translated to 3.3V). Active HIGH (HIGH means read). Is opposite from WR line. |
| WR | Write | GB write line (down-translated to 3.3V). Active LOW (LOW means write). Is opposite from RD line. |
| CLK | Clock | GB clock line (down-translated to 3.3V). Used to time read/write operations. |
| OE | Output Enable | Enable the output of data on the 64Mbit RAM chip. Pull high to tristate the output (for manipulating address lines via MCU). |
| CE | Chip Enable | Enable 64Mbit RAM chip. |
| WE | Write Enable | Specify a read or write operation on the 64Mbit RAM chip. |
| A22-14 | Address lines | Higher address lines of the 64Mbit RAM chip. |
| A15-0 | Address lines | GB address lines (down-translated to 3.3V) |

Some of the pins on the chip have either been pulled high or low, because they weren’t needed to be changed during operation. To identify these pins and see where they are tied to, please refer to the schematics. For more details about how to use these chips, please refer to their respected datasheets. To identify the pins, please refer to the silk screen on the board (or see picture below).

  
Image of top of board design, on which the silk screen can be visible.

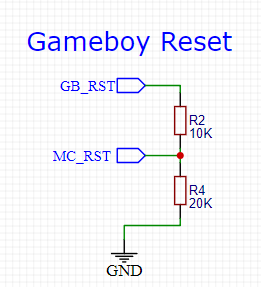
For more details on wiring, see schematics and/or board design.

### Fixes for next iteration

This board has a lot of faults. The faults are so big and numerous in fact, that I haven’t used this iteration for any testing. When I was inspecting the boards after I’d ordered them my big glowing brain started working again and I found a lot of things wrong with it (like missing pins and wrong connections). Nothing wrong with the company who made the board or the manufacture of the board, just design faults I made (to make things clear). Here’s a list of faults with the board to fix with the next iteration:

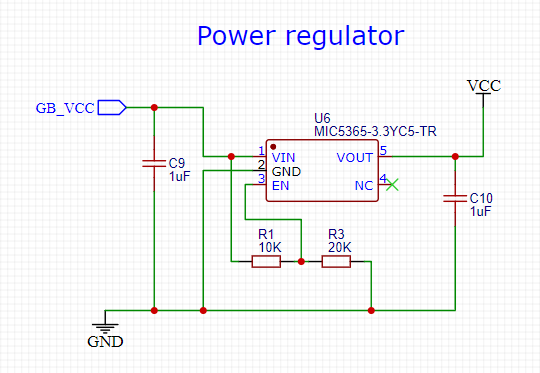
**RST Buffer doesn’t block 5V**When I designed the board, I wanted to be able to reset the GB via the RESET line. Because this is connected to the GB, it is 5V logic. To protect the MCU running on 3.3V, I put a buffer on the board connected to the RESET line (see U5). I thought this would block the 5V of the pin, but when not connecting the input of the buffer, the input sits at around 5V. When driving the pin to 3.3V (which would mean that the GB shouldn’t reset), it drops down to about 4V.

As this is not ideal, I’ve decided to remove the buffer and replace it with a voltage divider circuit. If I use a 10k and 20k resistor, I should be able to lower the RESET line voltage going to a GPIO pin on the MCU, whilst being able to pull this line down via the resistor (and resetting the GB).

  
Image showing current setup for the reset pin

**Linear power supply not enabled**I put a linear power supply on the board, in order to power the 3.3V components on the board from the 5V power supply coming from the GB. The chip I chose, had a enable pin, which was active high (the power supply is enabled when connected to HIGH). Because I apparently only had one braincell working, I connected it to the output of the supply. Because of this, you have a classic chicken and egg problem, where the enable line must go HIGH, but to do that, the supply must be enabled. Just wonderful…

The solution is simple, just connect the enable pin to a voltage divider. If I use a 10k and 20k resistor connected to 5V (GB power rail), the supply should drop to about 3V, which the enable pin sees as HIGH (according to the datasheet, it sees voltages higher than 1.2V as HIGH).

Image showing current setup for Linear power supply.

**Some dimensions are wrong**To get the dimensions of a board that could fit inside a cartridge, I using an empty plastic cartridge and the board inside it. I must have measured things almost perfectly, because the board was a tad to big (like maybe 0.2 bigger in most dimensions). Because of this, I had to sand some sides down to get it to fit.

Also, the thickness of the board it to large. When I tried to put the board inside the plastic casing, the lid wouldn’t close. I eventually got it closed, but then the card wouldn’t slide completely inside the cart slot. The board is to thicc to fit inside the cart slot.

And finally, the hole is a bit higher then it should be. Luckily, the hole was big enough to still fit, but this is not ideal.

I’ll give the next iteration some clearance, so that I don’t have to sand it again (not that it’s terrible, but since I’m making a new one anyway, I might as well change it). And more importantly, I next need to order a thinner PCB. I measured with a random board it to be 0.8 mm. Also, I need to change the size and position of the hole to fit nicely.

**Data bus not accessible via pin to MCU**Quite an embarrassing thing to forget, but I forgot to add pins to connect the data bus coming from the bus transceiver to the MCU. Because of this, the MCU can’t put data on the bus to write to memory, meaning it can’t load the game.

As this is pretty important, this should be fixed in the next iteration. I’ll next add some extra pins, which are connected to the data bus.

## Model PM - Prototype Modular

For the next iteration, I wanted to make a drastic change. When buying the parts for the last iteration, I accidentally bought the wrong 64Mbit chips. I originally ordered it from another supplier, because the SRAM chip costs about €26 on Mouser and I found another for about €2,50. The price difference should’ve been obvious, but I don’t think I had slept well that night. So now I have 256kbit chips laying around for other projects.

Because I don’t want to ‘waste’ a €26 chip on every iteration, I decided to make the next iteration modular. I made this one with the possibility to connect 2 daughter boards: one for memory and one for the MCU. This way I can reuse this iteration and when I want to add the MCU, I can just make a daughter board for it. For now, I’ll just make a daughter board with pins to connect to an external MCU (for testing purposes) and a daughter board for the memory (“ROM” and RAM).

### Hardware

Firstly, let’s go over some changes compared to the last iteration:

As stated in the introduction, I’ve made the change to be able to add daughterboards. I’ve opted for a 40 (2 x 20) pin interface with the memory board and a 50 (24 + 26) pin interface with the MCU board. I chose to use 2 connectors to distribute the pins better and to have a more stable physical connection.

The wiring of the address bus is slightly different. In the last iteration, I wired the address buss coming from the GB directly to both memory chips, while 2 separate busses for the higher addresses exist. Because the bus can either be used to communicate with the “ROM” chip or with the RAM chip, I’ve combined these 2 busses. Now, the bus is hilariously named the “Bank bus” used to select the wanted bank in memory. This saved me 4 pins, so quite nice. See diagram below for a visual representation. The control pins have also been changed, combining the WE (write enable) and OE (output enable) pins with separate chip select pins to select either chip, because again only one chip can be used at a time. **Warning: be sure to select only 1 chip at any time. Not doing so may damage them.**

**TODO: Add address diagram**

This iteration has the possibility to add a retainer for a coin cell battery. The battery can then be used for stuff like RTC time keeping and retaining data. The retainer I chose can hold 1220 and 1225 series batteries. I also included a charger circuit for the use of rechargeable batteries. This circuit can also be left out by not soldering the components, but instead shorting R5. I meant for the rail to be powered by 3V, however as the battery power pin is a separate pin going to both of the daughter boards, you could put any voltage on it (as long as the connected hardware support it).

Control over the transceivers has also been improved. This time I thought more about how both the GB and MCU use the parallel address and data bus and changed the operation of the transceivers in a way that this goal can be achieved.

The address transceiver only needs to pass data from the GB to the cart address bus (connected to MBC/MCU and RAM chips), so the DIR pin is still tied to ground. The transceiver has an additional pin that controls the pin state (when low, one bus is input and other output and when high, both busses are tri-stated). Originally I had this also tied to ground, but it’s something very handy for the MCU to control, so I connected this pin to the MCU interface (also for the data transceiver).

This enables the MCU to block signals coming from the GB and communicate with the RAM chips. This is useful, for example, if you want the MCU to run code on its own and use part of the RAM, creating shared RAM. When the GB is then not using the bus (in between read/writes) the MCU can use it to, for example, output data when being used as a co-processor of sorts. This level of control is then necessary, because when the GB is not communicating with the cart, it still pollutes the address and data bus.

Care should be taken when doing this, because the GB obviously loads instructions via the cart, so it uses this bus a lot. Luckily, it seems the communication is so slow (GB\_CLK works at 2MHz) that it should be possible to preform reads and writes in between the requests from the GB. The GB takes 480 ns preparing for the read/write and then an additional 480 ns to preform it. As the access time for both RAM chips is only 55 ns, you could (theoretically) preform 3 additional write/read requests when the GB is doing it’s thing.

**TODO: More research/prototyping in necessary for exact workings, but it should be possible.**