GBDK 2020 Docs

Generated by Doxygen 1.8.17

Sat Mar 20 2021 19:52:49

1 General Documentation

1.1 Introduction

Welcome to GBDK-2020! The best thing to do is head over to the Getting Started section to get up and running.

1.2 About the Documentation

This documentation is partially based on material written by the original GBDK authors in 1999 and updated for GBDK-2020. The API docs are automatically generated from the C header files using Doxygen.

GBDK-2020 is an updated version of the original GBDK with a modernized SDCC toolchain and many API improvements and fixes. It can be found at: `https://github.com/Zal0/gbdk-2020/`.

The original GBDK sources, documentation and website are at: `http://gbdk.sourceforge.net/`

1.3 About GBDK

The GameBoy Developer's Kit (GBDK, GBDK-2020) is used to develop games and programs for the Nintendo Game Boy system in C and assembly. GBDK includes a set of libraries for the most common requirements and generates image files for use with a real GameBoy or with emulators.

GBDK features:

- C and ASM toolchain based on SDCC with some support utilities
- A set of libraries with source code
- Example programs in ASM and in C
- Support for multiple ROM bank images

GBDK is freeware. Most of the tooling code is under the GPL. The runtime libraries should be under the LGPL. Please consider mentioning GBDK in the credits of projects made with it.

1.4 Historical Info and Links

The following is from the original GBDK documenation.
Thanks to quang for many of the comments to the gb functions. Some of the comments are ripped directly from the Linux Programmers manual, and some directly from the pan/k00Pa document.
quangDX.com
The (original) gbdk homepage
Jeff Frohwein's GB development page. A extensive source of Game Boy related information, including GeeBee's GB faq and the pan/k00Pa document.

2 Getting Started

Follow the steps in this section to start using GBDK-2020.

2.1 1. Compile Example projects

Make sure your GBDK-2020 installation is working correctly by compiling some of the included example projects. Navigate to the example projects folder ("examples/gb/" under your GBDK-2020 install folder) and open a command line. Then type:

```
make
```

This should build all of the examples sequentially. You can also navigate into an individual example project's folder and build it by typing make.
If everything works and there are no errors reported each example sub-folder should have it's on .gb ROM file.

2.2 2. Use a Template

To create a new project use a template!
There are template projects included in the GBDK example projects to help you get up and running. Their folder names start with template_.

1. Copy one of the template folders to a new folder name
2. If you moved the folder out of the GBDK examples then you **must** update the GBDK path variable and/or the path to LCC in the Makefile or make.bat so that it will still build correctly.
3. Type make on the command line in that folder to verify it still builds.
4. Open main.c to start making changes.

2.3 3. If you use GBTD / GBMB, get the fixed version

If you plan to use GBTD / GBMB for making graphics, make sure to get the version with the const fix and other improvements. See const_gbtd_gbmb.

2.4 4. Review Coding Guidelines

Take a look at the coding guidelines, even if you have experience writing software for other platforms. There is important information to help you get good results and performance on the Game Boy.
If you haven't written programs in C before, check the C tutorials section.

2.5 5. Hardware and Resources

If you have a specific project in mind, consider what hardware want to target. It isn't something that has to be decided up front, but it can influence design and implementation.
What size will your game or program be?

- 32K Cart (no-MBC required)
- Larger than 32K (MBC required)

- See more details about ROM Banking and MBCs.

What hardware will it run on?

- Game Boy (& Game Boy Color)
- Game Boy Color only
- Game Boy & Super Game Boy
- See how to set the compatibility type in the cartridge header. Read more about hardware differences in the Pandocs

2.6 6. Set up C Source debugging

Tracking down problems in code is easier with a debugger. Emulicious has a debug adapter that provides C source debugging with GBDK-2020.

2.7 7. Try a GBDK Tutorial

You might want to start off with a guided GBDK tutorial from the GBDK Tutorials section.

- **Note:** Tutorials (or parts of them) may be based on the older GBDK from the 2000's before it was updated to be GBDK-2020. The general principals are all the same, but the setup and parts of the toolchain (compiler/etc) may be somewhat different and some links may be outdated (pointing to the old GBDK or old tools).

2.8 8. Read up!

- It is strongly encouraged to read more GBDK-2020 General Documentation.
- Learn about the Game Boy hardware by reading through the Pandocs technical reference.

2.9 9. Need help?

Check out the links for online community and support and read the FAQ.

3 Links and Third-Party Tools

This is a brief list of useful tools and information. It is not meant to be complete or exhaustive, for a larger list see awesome_gb

3.1 SDCC Compiler Suite User Manual

- GBDK-2020 uses the SDCC compiler and related tools. The SDCC manual goes into much more detail about available features and how to use them.
  `http://sdcc.sourceforge.net/doc/sdccman.pdf`
  `http://sdcc.sourceforge.net`

3.2 Getting Help

- GBDK Discord community:
  `https://github.com/Zal0/gbdk-2020/#discord-servers`
- Game Boy discussion forum:
  `https://gbdev.gg8.se/forums/`

3.3 Game Boy Documentation

- **Pandocs**
  Extensive and up-to-date technical documentation about the Game Boy and related hardware.
  https://gbdev.io/pandocs/
- **Awesome Game Boy List**
  A list of Game Boy/Color development resources, tools, docs, related projects and homebrew.
  https://gbdev.io/list.html

3.4 Tutorials

- **Gaming Monsters Tutorials**
  Several video tutorials and code for making games with GBDK/GBDK-2020.
  https://www.youtube.com/playlist?list=PLeEj4c2zF7PaFv5MPYhNAkBGrkx4iPGJo
  https://github.com/gingemonster/GamingMonstersGameBoySampleCode

3.5 Example code

- **Simplified GBDK examples**
  https://github.com/mrombout/gbdk_playground/commits/master

3.6 Graphics Tools

- **Game Boy Tile Designer and Map Builder (GBTD / GBMB)**
  Sprite / Tile editor and Map Builder that can export to C that works with GBDK.
  - **Use this updated version:** (has const export fixed and other improvments):
    https://github.com/untoxa/GBTD_GBMB
  - This older version is **not recommended**:
    http://www.devrs.com/gb/hmgd/intro.html (old, original tools)
  - A GIMP plugin for import/export:
    https://github.com/bbbbbr/gimp-tilemap-gb (GIMP plugin to read/write GBR/GBM files)
- **Tilemap Studio**
  A tilemap editor for Game Boy, GBC, GBA, or SNES projects.
  https://github.com/Rangi42/tilemap-studio/

3.7 Music drivers and tools

- **GBT Player**
  A .mod converter and music driver that works with GBDK and RGBDS.
  https://github.com/AntonioND/gbt-player
  Docs from GBStudio that should mostly apply: https://www.gbstudio.dev/docs/music/
- **hUGEdriver**
  A tracker and music driver that works with GBDK and RGBDS. It is smaller, more efficient and more versatile than gbt_player.
  https://github.com/untoxa/hUGEBuild
  https://github.com/SuperDisk/hUGEDriver
  https://github.com/SuperDisk/hUGETracker

3.8 Emulators

- **BGB**
  Accurate emulator, has useful debugging tools.
  http://bgb.bircd.org/

- **Emulicious**
  An accurate emulator with extensive tools including source level debugging.
  https://emulicious.net/

3.9 Debugging tools

- **Emulicious debug adapter**
  Provides source-level debugging in VS Code that works with GBDK2020.
  https://marketplace.visualstudio.com/items?itemName=emulicious.emulicious-debugger

- **romusage**
  Calculate used and free space in banks (ROM/RAM) and warn about errors such as bank overflows.
  https://github.com/bbbbbr/romusage

- **noi2sym.py**
  Convert .noi files into a symbol format compatible with BGB. Allows BGB to recognize variables and functions based on address.
  https://github.com/untoxa/hUGEBuild/blob/master/tools/noi2sym.py

- **src2sym.pl**
  Add line-by-line C source code to the main symbol file in a BGB compatible format. This allows for C source-like debugging in BGB in a limited way. https://gbdev.gg8.se/forums/viewtopic.↩
  php?id=710

4 Using GBDK

4.1 Interrupts

Interrupts allow execution to jump to a different part of your code as soon as an external event occurs - for example the LCD entering the vertical blank period, serial data arriving or the timer reaching its end count. For an example see the irq.c sample project.
Interrupts in GBDK are handled using the functions disable_interrupts(), enable_interrupts(), set_interrupts(UBYTE ier) and the interrupt service routine (ISR) linkers add_VBL(), add_TIM, add_LCD, add_SIO and add_JOY which add interrupt handlers for the vertical blank, timer, LCD, serial link and joypad interrupts respectively.
Since an interrupt can occur at any time an Interrupt Service Request (ISR) cannot take any arguments or return anything. Its only way of communicating with the greater program is through the global variables. When interacting with those shared ISR global variables from main code outside the interrupt, it is a good idea to wrap them in a `critical {}` section in case the interrupt occurs and modifies the variable while it is being used.
Interrupts should be disabled before adding ISRs. To use multiple interrupts, *logical OR* the relevant IFLAGs together.
ISRs should be kept as small and short as possible, do not write an ISR so long that the Game Boy hardware spends all of its time servicing interrupts and has no time spare for the main code.
For more detail on the Game Boy interrupts consider reading about them in the Pandocs.

4.1.1 Available Interrupts

The GameBoy hardware can generate 5 types of interrupts. Custom Interrupt Service Routines (ISRs) can be added in addition to the built-in ones available in GBDK.

- VBL : LCD Vertical Blanking period start
  - The default VBL ISR is installed automatically.
    - See add_VBL() and remove_VBL()

- LCD : LCDC status (such as the start of a horizontal line)
  - See add_LCD() and remove_LCD()
  - Example project: `lcd_isr_wobble`
- TIM : Timer overflow
  - See add_TIM() and remove_TIM()
  - Example project: `tim`
- SIO : Serial Link I/O transfer end
  - The default SIO ISR gets installed automatically if any of the standard SIO calls are used. These calls include add_SIO(), remove_SIO(), send_byte(), receive_byte().
  - The default SIO ISR cannot be removed once installed. Only secondary chained SIO ISRs (added with add_SIO()) can be removed.
  - See add_SIO() and remove_SIO()
  - Example project: `comm`
- JOY : Transition from high to low of a joypad button
  - See add_JOY() and remove_JOY()

4.1.2 Adding your own interrupt handler

It is possible to install your own interrupt handlers (in C or in assembly) for any of these interrupts. Up to 4 chained handlers may be added, with the last added being called last. If the remove_VBL() function is to be called, only three may be added for VBL.
Interrupt handlers are called in sequence. To install a new interrupt handler, do the following:

1. Write a function (say foo()) that takes no parameters, and that returns nothing. Remember that the code executed in an interrupt handler must be short.
2. Inside a `__critical { ... }` section, install your interrupt handling routines using the add_XXX() function, where XXX is the interrupt that you want to handle.
3. Enable interrupts for the IRQ you want to handle, using the set_interrupts() function. Note that the VBL interrupt is already enabled before the main() function is called. If you want to set the interrupts before main() is called, you must install an initialization routine.

See the `irq` example project for additional details for a complete example.

4.1.3 Returning from Interrupts and STAT mode

By default when an Interrupt handler completes and is ready to exit it will check STAT_REG and only return at the BEGINNING of either LCD Mode 0 or Mode 1. This helps prevent graphical glitches caused when an ISR interrupts a graphics operation in one mode but returns in a different mode for which that graphics operation is not allowed.
You can change this behavior using nowait_int_handler() which does not check STAT_REG before returning. Also see wait_int_handler().

4.2 What GBDK does automatically and behind the scenes

4.2.1 OAM (VRAM Sprite Attribute Table)

GBDK sets up a Shadow OAM which gets copied automatically to the hardware OAM by the default V-Blank ISR. The Shadow OAM allows updating sprites without worrying about whether it is safe to write to them or not based on the hardware LCD mode.

4.2.2 Font tiles when using stdio.h

Including stdio.h and using functions such as printf() will use a large number of the background tiles for font characters. If stdio.h is not included then that space will be available for use with other tiles instead.

4.2.3 Default Interrupt Service Handlers (ISRs)

- V-Blank: A default V-Blank ISR is installed on startup which copies the Shadow OAM to the hardware OAM and increments the global sys_time variable once per frame.
- Serial Link I/O: If any of the GBDK serial link functions are used such as send_byte() and receive_byte(), the default SIO serial link handler will be installed automatically at compile-time.

4.3 Copying Functions to RAM and HIRAM

The `ram_function` example project included with GBDK demonstrates copying functions to RAM and HIRAM.
It is possible to copy functions to RAM and HIRAM (using the memcpy() and hiramcpy() functions), and execute them from C. The compiler automatically generates two symbols for the start and the end of each function, named start_X and end_X (where X is the name of the function). This enables to calculate the length of a function when copying it to RAM. Ensure you have enough free space in RAM or HIRAM for copying a function.
There are basically two ways for calling a function located in RAM, HIRAM, or ROM:

- Declare a pointer-to-function variable, and set it to the address of the function to call.
- Declare the function as extern, and set its address at link time using the -Wl-gXXX=# flag (where XXX is the name of the function, and # is its address).

The second approach is slightly more efficient. Both approaches are demonstrated in the `ram_function.c` example.

4.4 Mixing C and Assembly

You can mix C and assembly (ASM) in two ways as described below. For additional detail see the links_sdcc_docs.

4.4.1 Inline ASM within C source files

Example:

```
__asm__("nop");
```

Another Example:

```
void some_c_function()
{
  // Optionally do something
  __asm
      (ASM code goes here)
  __endasm;
}
```

4.4.2 In Separate ASM files

Todo This is from GBDK 2.x docs, verify it with GBDK-2020 and modern SDCC

It is possible to assemble and link files written in ASM alongside files written in C.

- A C identifier `i` will be called `_i` in assembly.
- Results are always returned into the `DE` register.
- Parameters are passed on the stack (starting at `SP+2` because the return address is also saved on the stack).
- Assembly identifier are exported using the `.globl` directive.
- You can access GameBoy hardware registers using `_reg_0xXX` where `XX` is the register number (see `sound.c` for an example).
- Registers must be preserved across function calls (you must store them at function begin, and restore them at the end), except `HL` (and `DE` when the function returns a result).

Here is an example of how to mix assembly with C:
main.c

```
main()
{
  WORD i;
  WORD add(WORD, WORD);

  i = add(1, 3);
}
```

add.s

```
.globl _add
_add:          ; WORD add(WORD a, WORD b)
               ; There is no register to save:
               ;  BC is not used
               ;  DE is the return register
               ;  HL needs never to be saved
LDA   HL,2(SP)
LD    E,(HL)   ; Get a in DE
INC   HL
LD    D,(HL)
INC   HL
LD    A,(HL)   ; Get b in HL
INC   HL
LD    H,(HL)
LD    L,A
ADD   HL,DE    ; Add DE to HL
LD    D,H
LD    E,L
               ; There is no register to restore
RET            ; Return result in DE
```

4.5 Known Issues and Limitations

4.5.1 SDCC

- Const arrays declared with `somevar[n] = {x}` will **NOT** get initialized with value `x`. This may change when the SDCC RLE initializer is fixed. Use memset for now if you need it.
- SDCC banked calls and far_pointers in GBDK only save one byte for the ROM bank, so for example they are limtied to **bank 15** max for MBC1 and **bank 255** max for MBC5. See banked_calls for more details.

5 Coding Guidelines

5.1 Learning C / C fundamentals

Writing games and other programs with GBDK will be much easier with a basic understanding of the C language. In particular, understanding how to use C on "Embedded Platforms" (small computing systems, such as the Game Boy) can help you write better code (smaller, faster, less error prone) and avoid common pitfals.

5.1.1 General C tutorials

- https://www.learn-c.org/
- https://www.tutorialspoint.com/cprogramming/index.htm

5.1.2 Embedded C introductions

- http://dsp-book.narod.ru/CPES.pdf
- https://www.phaedsys.com/principals/bytecraft/bytecraftdata/bcfirststeps.pdf

5.1.3 Game Boy games in C

- https://gbdev.io/list.html#c

5.2 Understanding the hardware

In addition to understanding the C language it's important to learn how the Game Boy hardware works. What it is capable of doing, what it isn't able to do, and what resources are available to work with. A good way to do this is by reading the Pandocs and checking out the awesome_gb list.

5.3 Writing optimal C code for the Game Boy and SDCC

The following guidelines can result in better code for the Game Boy, even though some of the guidance may be contrary to typical advice for general purpose computers that have more resources and speed.

5.3.1 Tools

5.3.1.1 GBTD / GBMB, Arrays and the "const" keyword **Important**: The old GBTD/GBMB fails to include the `const` keyword when exporting to C source files for GBDK. That causes arrays to be created in RAM instead of ROM, which wastes RAM, uses a lot of ROM to initialize the RAM arrays and slows the compiler down a lot.
__Use of toxa's updated GBTD/GBMB is highly recommended.__
If you wish to use the original tools, you must add the `const` keyword every time the graphics are re-exported to C source files.

5.3.2 Variables

- Use 8-bit values as much as possible. They will be much more efficient and compact than 16 and 32 bit types.
- Prefer unsigned variables to signed ones: The code generated will be generally more efficient, espacially when comparing two values.
- Use explicit types so you always know the size of your variables. `INT8`, `UINT8`, `INT16`, `UINT16`, `INT32`, `UINT32` or `BYTE`, `UBYTE`, `WORD`, `UWORD`, `LWORD`, `ULWORD`.
  Types are defined in asm/types.h and asm/gbz80/types.h
- Global and local static variables are generally more efficient than local non-static variables (which go on the stack and are slower and can result in slower code).
- `const` keyword: Use const for arrays, structs and variables with read-only (constant) data. It will reduce ROM, RAM and CPU usage significantly. Non-`const` values are loaded from ROM into RAM inefficiently, and there is no benefit in loading them into the limited available RAM if they aren't going to be changed.
- For calculated values that don't change, pre-compute results once and store the result. Using lookup-tables and the like can improve speed and reduce code size. Macros can sometimes help. It may be beneficial to do the calculations with an outside tool and then include the result as C code in a const array.
- Use an advancing pointer (`someStruct->var = x; someStruct++`) to loop through arrays of structs instead of using indexing each time in the loop `someStruct[i].var = x`.
- When modifying variables that are also changed in an Interrupt Service Routine (ISR), wrap them the relevant code block in a `__critical { }` block. See http://sdcc.sourceforge.net/doc/sdccman.pdf#section.3.9
- When using constants and literals the `U`, `L` and `UL` postfixes can be used.
  - `U` specifies that the constant is unsigned
  - `L` specifies that the constant is long.
  - NOTE: In SDCC 3.6.0, the default for char changed from signed to unsigned. The manual says to use `--fsigned-char` for the old behavior, this option flag is included by default when compiling through lcc.

5.3.3 Code structure

- When procesing for a given frame is done and it is time to wait before starting the next frame, wait_vbl_done() can be used. It uses HALT to put the CPU into a low power state until processing resumes. The CPU will wake up and resume processing at the end of the current frame when the Vertical Blanking interrupt is triggered.
- Minimize use of multiplication, modulo and division. These operations have no corresponding CPU instructions (software functions), and hence are time costly. Division by powers of 2 are better, they have specific SDCC optimizations.
  - Alternatives to modulo:
    * When using power of 2 you can use bit masks. Example: `(n % 8)` can be achieved with `(n & 0x7)`
    * If you need decimal numbers to count or display a score, you can use the GBDK BCD (`binary coded decimal`) number functions. See: bcd.h and the `BCD` example project included with GBDK.
- Avoid long lists of function parameters. Passing many parameters can add overhead, especially if the function is called often. When applicable globals and local static vars can be used instead.
- Use inline functions if the function is short. (with the `inline` keyword, such as `inline UINT8 myFunction() { ... }`)
- Do not use recursive functions
- Prefer == and != comparison operators to <, <=, >, and >=. The code will be shorter and quicker.

  It is even faster to check if a variable is 0 than if it is equal to some other value, so looping from *N down to zero* is faster than looping *from zero up to N*.

For instance:

```
for(i = 0; i < 10; i++)
```

is less efficient than:

```
for(i = 0; i != 10; i++)
```

and if possible, even better:

```
for(i = 10; i != 0; i--)
```

5.3.4 GBDK API/Library

- stdio.h: If you have other ways of printing text, avoid including stdio.h and using functions such as printf(). Including it will use a large number of the background tiles for font characters. If stdio.h is not included then that space will be available for use with other tiles instead.
- drawing.h: The Game Boy graphics hardware is not well suited to frame-buffer style graphics such as the kind provided in drawing.h. Due to that, most drawing functions (rectangles, circles, etc) will be slow . When possible it's much faster and more efficient to work with the tiles and tile maps that the Game Boy hardware is built around.
- waitpad() and waitpadup check for input in a loop that doesn't HALT at all, so the CPU will be maxed out until it returns. One alternative is to write a function with a loop that checks input with joypad() and then waits a frame using wait_vbl_done() (which idles the CPU while waiting) before checking input again.

5.3.5 Toolchain

- See SDCC optimizations: `http://sdcc.sourceforge.net/doc/sdccman.pdf#section.8.1`
- Use profiling. Look at the ASM generated by the compiler, write several versions of a function, compare them and choose the faster one.
- Use the SDCC `--max-allocs-per-node` flag with large values, such as `50000`. `--opt-code-speed` has a much smaller effect.

  - GBDK-2020 (after v4.0.1) compiles the library with `--max-allocs-per-node 50000`, but it must be turned on for your own code.
    (example: `lcc ... -Wf--max-allocs-per-mode 50000` or `sdcc ... --max-allocs-per-mode 50000`).
  - The other code/speed flags are `--opt-code-speed` or `--opt-code-size`.

- Use current SDCC builds from http://sdcc.sourceforge.net/snap.php
  The minimum required version of SDCC will depend on the GBDK-2020 release. See GBDK Releases

- Learn some ASM and inspect the compiler output to understand what the compiler is doing and how your code gets translated. This can help with writing better C code and with debugging.

5.3.6 chars and vararg functions

In standard C when `chars` are passed to a function with variadic arguments (varargs, those delcared with `...` as a parameter), such as printf(), those `chars` get automatically promoted to `ints`. For an 8 bit cpu such as the Game Boy's, this is not as efficient or desireable in most cases. So the default SDCC behavior, which GBDK-2020 expects, is that chars will remain chars and *not* get promoted to ints when **explicitly cast as chars while calling a varargs function**.

- They must be explicitly re-cast when passing them to a varargs function, even though they are already declared as chars.

- Discussion in SDCC manual:
  http://sdcc.sourceforge.net/doc/sdccman.pdf#section.1.5
  http://sdcc.sourceforge.net/doc/sdccman.pdf#subsection.3.5.10

- If SDCC is invoked with -std-cxx (–std-c89, –std-c99, –std-c11, etc) then it will conform to standard C behavior and calling functions such as printf() with chars may not work as expected.

For example:

```
unsigned char i = 0x5A;

// NO:
// The char will get promoted to an int, producing incorrect printf output
// The output will be: 5A 00
printf("%hx %hx", i, i);

// YES:
// The char will remain a char and printf output will be as expected
// The output will be: 5A 5A
printf("%hx %hx", (unsigned char)i, (unsigned char)i);
```

Some functions that accept varargs:

- BGB_MESSAGE_FMT, gprintf(), printf(), sprintf()

Also See:

- Other cases of char to int promotion: http://sdcc.sourceforge.net/doc/sdccman.pdf#chapter.6

5.4 When C isn't fast enough

Todo Update and verify this section for the modernized SDCC and toolchain

For many applications C is fast enough but in intensive functions are sometimes better written in assembler. This section deals with interfacing your core C program with fast assembly sub routines.

5.4.1 Calling convention

sdcc in common with almost all C compilers prepends a '_' to any function names. For example the function printf(...) begins at the label _printf::. Note that all functions are declared global.
The parameters to a function are pushed in right to left order with no aligning - so a byte takes up a byte on the stack instead of the more natural word. So for example the function int store_byte(UWORD addr, UBYTE byte) would push 'byte' onto the stack first then addr using a total of three bytes. As the return address is also pushed, the stack would contain:

```
At SP+0 - the return address

At SP+2 - addr

At SP+4 - byte
```

Note that the arguments that are pushed first are highest in the stack due to how the Game Boy's stack grows downwards.
The function returns in DE.

5.4.2 Variables and registers

C normally expects registers to be preserved across a function call. However in the case above as DE is used as the return value and HL is used for anything, only BC needs to be preserved.
Getting at C variables is slightly tricky due to how local variables are allocated on the stack. However you shouldn't be using the local variables of a calling function in any case. Global variables can be accessed by name by adding an underscore.

5.4.3 Segments

The use of segments for code, data and variables is more noticeable in assembler. GBDK and SDCC define a number of default segments - `_CODE`, `_DATA` and `_BSS`. Two extra segments `_HEADER` and `_HEAP` exist for the Game Boy header and malloc heap respectively.
The order these segments are linked together is determined by crt0.s and is currently `_CODE` in ROM, then `_DATA`, `_BSS`, `_HEAP` in WRAM, with `STACK` at the top of WRAM. `_HEAP` is placed after `_BSS` so that all spare memory is available for the malloc routines. To place code in other than the first two banks, use the segments `_CODE_x` where x is the 16kB bank number.
As the `_BSS` segment occurs outside the ROM area you can only use .ds to reserve space in it.
While you don't have to use the `_CODE` and `_DATA` distinctions in assembler you may wish to do so consistancy.

6 ROM/RAM Banking and MBCs

6.1 ROM/RAM Banking and MBCs (Memory Bank Controllers)

The standard Game Boy cartridge with no MBC has a fixed 32K bytes of ROM. In order to make cartridges with larger ROM sizes (to store more code and graphics) MBCs can be used. They allow switching between multiple ROM banks that use the same memory region. Only one of the banks can be selected as active at a given time, while all the other banks are inactive (and so, inaccessible).

6.1.1 Unbanked cartridges

Cartridges with no MBC controller are unbanked, they have 32K bytes of fixed ROM space and no switchable banks. For these cartridges the ROM space between `0000h and 7FFFh` can be treated as a single large bank of 32K bytes, or as two contiguous banks of 16K bytes in Bank 0 at `0000h - 3FFFh` and Bank 1 at `4000h to 7FFFh`.

6.1.2 MBC Banked cartridges (Memory Bank Controllers)

Cartridges with MBCs allow the the Game Boy to work with ROMS up to 8MB in size and with RAM up to 128kB. Each bank is 16K Bytes.

- Bank 0 of the ROM is located in the region at `0000h - 3FFFh`. It is *usually* fixed (unbanked) and cannot be switched out for another bank.

- The higher region at `4000h to 7FFFh` is used for switching between different ROM banks.

See the Pandocs for more details about the individual MBCs and their capabilities.

6.2 Working with Banks

To assign code and constant data (such as graphics) to a ROM bank and use it:

- Place the code for your ROM bank in one or several source files.
- Specify the ROM bank to use, either in the source file or at compile/link time.
- Specify the number of banks and MBC type during link time.
- When the program is running and wants to use data or call a function that is in a given bank, manually or automatically set the desired bank to active.

6.2.1 Setting the ROM bank for a Source file

The ROM and RAM bank for a source file can be set in a couple different ways. Multiple different banks cannot be assigned inside the same source file (unless the `___addressmod` method is used), but multiple source files can share the same bank.
If no ROM and RAM bank are speciied for a file then the default _CODE, _BSS and _DATA segments are used.
Ways to set the ROM bank for a Source file

- `#pragma bank <N>` at the start of a source file. Example (ROM bank 2): `#pragma bank 2`
- The lcc switch for ROM bank `-Wf-bo<N>`. Example (ROM bank 2): `-Wf-bo2`
- Using rom_autobanking

Note: You can use the `UNBANKED` keyword to define a function as unbanked if it resides in a source file which has been assigned a bank.

6.2.2 Setting the RAM bank for a Source file

- Using the lcc switch for RAM bank `-Wf-ba<N>`. Example (RAM bank 3): `-Wf-bo3`

6.2.3 Setting the MBC and number of ROM & RAM banks available

At the link stage this is done with lcc using pass-through switches for makebin.

- `-Wl-yo<N>` where `<N>` is the number of ROM banks. 2, 4, 8, 16, 32, 64, 128, 256, 512
- `-Wl-ya<N>` where `<N>` is the number of RAM banks. 2, 4, 8, 16, 32
- `-Wl-yt<N>` where `<N>` is the type of MBC cartridge (see below).

The following MBC settings are available when using the makebin MBC switch.

```
# From Makebin source:
#
#-Wl-yt<NN> where <NN> is one of the numbers below
#
# 0147: Cartridge type:
# 0-ROM ONLY             12-ROM+MBC3+RAM
# 1-ROM+MBC1             13-ROM+MBC3+RAM+BATT
# 2-ROM+MBC1+RAM         19-ROM+MBC5
# 3-ROM+MBC1+RAM+BATT    1A-ROM+MBC5+RAM
# 5-ROM+MBC2             1B-ROM+MBC5+RAM+BATT
# 6-ROM+MBC2+BATTERY     1C-ROM+MBC5+RUMBLE
# 8-ROM+RAM              1D-ROM+MBC5+RUMBLE+SRAM
# 9-ROM+RAM+BATTERY      1E-ROM+MBC5+RUMBLE+SRAM+BATT
# B-ROM+MMM01            1F-Pocket Camera
# C-ROM+MMM01+SRAM       FD-Bandai TAMA5
# D-ROM+MMM01+SRAM+BATT FE - Hudson HuC-3
# F-ROM+MBC3+TIMER+BATT FF - Hudson HuC-1
# 10-ROM+MBC3+TIMER+RAM+BATT
# 11-ROM+MBC3
```

6.2.4 Banked Functions

Banked functions can be called as follows.

- When defined with the `BANKED` keyword. Example: `void my_function() BANKED { do stuff }` in a source file which has had it's bank set (see above).
- Using far_pointers
- When defined with an area set up using the `__addressmod` keyword (See the `banks_new` example project and the SDCC manual for details)
- Using SWITCH_ROM_MBC1() (and related functions for other MBCs) to manually switch in the required bank and then call the function.

Unbanked functions (either in fixed Bank 0, or in an Unbanked ROM with no MBC)

- May call functions in any bank: **YES**
- May use data in any bank: **YES**

Todo Fill in this info for Banked Functions Banked functions (located in a switchable ROM bank)

- May call functions in any bank: ?
- May use data in any bank: **NO** (may only use data from currently active banks)

Limitations:

- SDCC banked calls and far_pointers in GBDK only save one byte for the ROM bank. So, for example, they are limtied to **bank 31** max for MBC1 and **bank 255** max for MBC5. This is due to the bank switching for those MBCs requiring a second, additional write to select the upper bits for more banks (banks 32+ in MBC1 and banks 256+ in MBC5).

6.2.5 Const Data (Variables in ROM)

Todo Const Data (Variables in ROM)

6.2.6 Variables in RAM

Todo Variables in RAM

6.2.7 Far Pointers

Far pointers include a segment (bank) selector so they are able to point to addresses (functions or data) outside of the current bank (unlike normal pointers which are not bank-aware). A set of macros is provided by GBDK 2020 for working with far pointers.
Warning: Do not call the far pointer function macros from inside interrupt routines (ISRs). The far pointer function macros use a global variable that would not get restored properly if a function called that way was interrupted by another one called the same way. However, they may be called recursively.
See FAR_CALL, TO_FAR_PTR and the `banks_farptr` example project.

6.2.8 Bank switching

You can manually switch banks using the SWITCH_ROM_MBC1(), SWITCH_RAM_MBC1(), and other related macros. See `banks.c` project for an example.
Note: You can only do a switch_rom_bank call from unbanked `_CODE` since otherwise you would switch out the code that was executing. Global routines that will be called without an expectation of bank switching should fit within the limited 16k of unbanked `_CODE`.

6.2.9 Restoring the current bank (after calling functions which change it without restoring)

If a function call is made (for example inside an ISR) which changes the bank *without* restoring it, then the _current_bank variable should be saved and then restored.
For example, **instead** of this code:

```
void vbl_music_isr(void)
{
    // A function which changes the bank and
    // *doesn't* restore it after changing.
    some_function();
}
```

It should be:

```
void vbl_music_isr(void)
{
    // Save the current bank
    UBYTE _saved_bank = _current_bank;
    // A function which changes the bank and
    // *doesn't* restore it after changing.
    some_function();
    // Now restore the current bank
    SWITCH_ROM_MBC5(_saved_bank);
}
```

6.2.10 Currently active bank: _current_bank

The global variable _current_bank is updated automatically when calling SWITCH_ROM_MBC1() and SWITCH_ROM_MBC5, or when a `BANKED` function is called.

6.3 Auto-Banking

A ROM bank auto-assignment feature was added in GBDK 2020 4.0.2.
Instead of having to manually specify which bank a source file will reside it, the banks can be assigned automatically to make the best use of space. The bank assignment operates on object files, after compiling/assembling and before linking.
To turn on auto-banking, use the `-autobank` argument with lcc
For a source example see the `banks_autobank` project.
In the source files you want auto-banked, do the following:

- Set the bank for the source file to `255`: `#pragma bank 255`
- Create a constant with no value to store the bank number for the source file: `const void __at(255) __bank_<name-for-a-given-source-file>;`.
  This constant can then be used for obtaining that files bank number with `(UINT8)&__bank_<name-for-a-given-source-file`.

Example: level_1_map.c

```
#pragma bank 255
const void __at(255) __bank_level_1_map;

const UINT8 my_level_1_map[] = {... some map data here ...};
```

Accessing that data: main.c

```
SWITCH_ROM_MBC1( (UINT8)&__bank_level_1_map );
// Do something with my_level_1_map[]
```

Features and Notes:

- Fixed banked source files can be used in the same project as auto-banked source files. The bankpack tool will attempt to pack the auto-banked source files as efficiently as possible around the fixed-bank ones.

Making sure bankpack checks all files:

- In order to correctly calculate the bank for all files every time, it is best to use the `-ext=` flag and save the auto-banked output to a different extension (such as `.rel`) and then pass the modified files to the linker. That way all object files will be processed each time the program is compiled.

```
Recommended:
.c and .s -> (compiler) .o -> (bankpack) -> .rel -> (linker) ... -> .gb
```

- It is important because when bankpack assigns a bank for an autobanked (bank=255) object file (.o) it rewrites the bank and will then no longer see the file as one that needs to be auto-banked. That file will then remain in it's previously assigned bank until a source change causes the compiler to rebuild it to an object file again which resets it's bank to 255.
- For example consider a fixed-bank source file growing too large to share a bank with an auto-banked source file that was previously assigned to it. To avoid a bank overflow it would be important to have the auto-banked file check every time whether it can share that bank or not.
- See bankpack for more options and settings

Limitations:

- At this time, the constant entries that get rewritten with the assigned bank (const void **at(255) __bank_**<name-you-want-to-use-for-that-source-file>**;) __cannot** be used from the source file they are declared in. In that case SDCC converts the bank number before bankpack has a chance to rewrite it. It may be referenced from any other source file, but not it's own.

6.4 Errors related to banking (overflow, multiple writes to same location)

A *bank overflow* during compile/link time (in makebin) is when more code and data are allocated to a ROM bank than it has capacity for. The address for any overflowed data will be incorrect and the data is potentially unreachable since it now resides at the start of a different bank instead of the end of the expected bank.
See the FAQ entry about bank overflow errors.
The current toolchain can only detect and warn (using ihxcheck) when one bank overflows into another bank that has data at its start. It cannot warn if a bank overflows into an empty one. For more complete detection, you can use the third-party romusage tool.

6.5 Bank space usage

In order to see how much space is used or remains available in a bank, you can use the third-party romusage tool.

6.5.1 Other important notes

- The SWITCH_ROM_MBC5 macro is not interrupt-safe. If using less than 256 banks you may always use SWITCH_ROM_MBC1 - that is faster. Even if you use mbc5 hardware chip in the cart.

6.6 Banking example projects

There are several projects in the GBDK 2020 examples folder which demonstrate different ways to use banking.

- `Banks`: A basic banking example
- `Banks_new`: Examples of using new bank assignment and calling conventions available in GBDK 2020 and it's updated SDCC version.
- `Banks_farptr`: Using far pointers which have the bank number built into the pointer.
- `Banks_autobank`: Shows how to use the bank auto-assignment feature of in GBDK 2020 4.0.2 or later, instead of having to manually specify which bank a source file will reside it.

7 GBDK Toolchain

7.1 Overview

GBDK 2020 uses the SDCC compiler along with some custom tools to build Game Boy ROMs.

- All tools are located under `bin/`
- The typical order of tools called is as follows. (When using lcc these steps are usually performed automatically.)

1. Compile and assemble source files (.c, .s, .asm) with sdcc and sdasgb
2. Optional: perform auto banking with bankpack on the object files
3. Link the object files into .ihx file with sdldgb
4. Validate the .ihx file with ihxcheck
5. Convert the .ihx file to a ROM file (.gb, .gbc) with makebin

To see individual arguments and options for a tool, run that tool from the command line with either no arguments or with `-h`.

7.2 Data Types

For data types and special C keywords, see asm/gbz80/types.h and asm/types.h.
Also see the SDCC manual (scroll down a little on the linked page): http://sdcc.sourceforge.net/doc/sdccman.pdf#section.1.1

7.3 Changing Important Addresses

It is possible to change some of the important addresses used by the toolchain at link time using the -Wl-g XXX=YYY and =Wl-b XXX=YYY flags (where XXX is the name of the data, and YYY is the new address).
lcc will include the following linker defaults for sdldgb if they are not defined by the user.

- `_shadow_OAM`
  - Location of sprite ram (requires 0xA0 bytes).
  - Default `-Wl-g _shadow_OAM=0xC000`
- `.STACK`
  - Initial stack address
  - Default `-Wl-g .STACK=0xE000`
- `.refresh_OAM`
  - Address to which the routine for refreshing OAM will be copied (must be in HIRAM). Default
  - Default `-Wl-g .refresh_OAM=0xFF80`
- `_DATA`
  - Start of RAM section (starts after Shadow OAM)
  - Default `-Wl-b _DATA=0xc0A0`
- `_CODE`
  - Start of ROM section
  - Default `-Wl-b _CODE=0x0200`

7.4 Compiling programs

The lcc program is the front end compiler driver for the actual compiler, assembler and linker. It works out what you want to do based on command line options and the extensions of the files you give it, computes the order in which the various programs must be called and then executes them in order. Some examples are:

- Compile the C source 'source.c', assemble and link it producing the Gameboy image 'image.gb'

```
lcc -o image.gb source.c
```

- Assemble the file 'source.s' and link it producing the Gameboy image 'image.gb'

```
lcc -o image.gb source.s
```

- Compile the C program 'source1.c' and assemble it producing the object file 'object1.o' for later linking.

```
lcc -c -o object1.o source1.c
```

- Assemble the file 'source2.s' producing the object file 'object2.o' for later linking

```
lcc -c -o object2.o source2.s
```

- Link the two object files 'object1.o' and 'object2.o' and produce the Gameboy image 'image.gb'

```
lcc -o image.gb object1.o object2.o
```

- Do all sorts of clever stuff by compiling then assembling source1.c, assembling source2.s and then linking them together to produce image.gb.

```
lcc -o image.gb source1.c source2.s
```

Arguments to the assembler etc can be passed via lcc using -Wp..., -Wf..., -Wa... and -Wl... to pass options to the pre-processor, compiler, assembler and linker respectivly. Some common options are:

- To generate an assembler listing file.

```
-Wa-l
```

- To generate a linker map file.

```
-Wl-m
```

- To bind var to address 'addr' at link time.

```
-Wl-gvar=addr
```

For example, to compile the example in the memory section and to generate a listing and map file you would use the following. Note the leading underscore that C adds to symbol names.

```
lcc -Wa-l -Wl-m -Wl-g_snd_stat=0xff26 -o image.gb hardware.c
```

7.4.1 Makefiles

Using Makefiles
Please see the sample projects included with GBDK-2020 for a couple different examples of how to use Makefiles. You may also want to read a tutorial on Makefiles. For example:
https://makefiletutorial.com/ https://www.tutorialspoint.com/makefile/index.↩
htm

7.5 Build Tools

7.5.1 lcc

lcc is the compiler driver (front end) for the GBDK/sdcc toolchain. It can be used to invoke all the tools needed for building a rom. If preferred, the individual tools can be called directly.

- the `-v` flag can be used to show the exact steps lcc executes for a build
- lcc can compile, link and generate a binary in a single pass: `lcc -o somerom.gb somesource.c`
- lcc now has a `-debug` flag that will turn on the following recommended flags for debugging
  - `--debug` for sdcc (lcc equiv: `-Wf-debug`)
  - `-y` enables .cdb output for sdldgb (lcc equiv: `-Wl-y`)
  - `-j` enables .noi output for sdldgb (lcc equiv: `-Wl-j`)

7.5.2 sdcc

SDCC C Source compiler

- Arguments can be passed to it through lcc using `-Wf-<argument>` and `-Wp-<argument>` (preprocessor)

7.5.3 sdasgb

SDCC Assembler for the gameboy

- Arguments can be passed to it through lcc using `-Wa-<argument>`

7.5.4 bankpack

Automatic Bank packer
When enabled, automatically assigns banks for object files where bank has been set to `255`, see rom_autobanking.
Unless an alternative output is specified the given object files are updated with the new bank numbers.

- Can be enabled by using the `-autobank` argument with lcc.
- Must be called after compiling/assembling and before linking
- Arguments can be passed to it through lcc using `-Wb-<argument>`

Limitations

- `__banked` functions cannot be called from within the same source file they are declared in.
- With data it is easier, because if you access data from the code in the same bank you don't need to switch the bank (access to `__bank_*` symbol).

7.5.5 sdldgb

The SDCC linker for the gameboy.
Links object files (.o) into a .ihx file which can be processed by makebin

- Arguments can be passed to it through lcc using `-Wl-<argument>`

7.5.6 ihxcheck

IHX file validator
Checks .ihx files produced by sdldgb for correctness.

- It will warn if there are multiple writes to the same ROM address. This may indicate mistakes in the code or ROM bank overflows
- Arguments can be passed to it through lcc using `-Wi-<argument>`

7.5.7 makebin

IHX to ROM converter
Converts .ihx files produced by sdldgb into ROM files (.gb, .gbc).

- Arguments can be passed to it through lcc using `-Wm-<argument>`

7.6 GBDK Utilities

7.6.1 GBCompress

Compresssion utility
Compresses (and decompresses) binary file data with the gbcompress algorithm (also used in GBTD/GBMB).
Decompression support is available in GBDK, see gb_decompress().

7.6.2 PNG to Metasprite

Tool for converting PNGs into GBDK format MetaSprites

Todo Document png2mtspr

8 Example Programs

GBDK includes several example programs both in C and in assembly. They are located in the examples directory, and in its subdirectories. They can be built by typing `make` in the correnponding directory.

8.1 banks (various projects)

There are several different projects showing how to use ROM banking with GBDK.

8.2 comm

Illustrates how to use communication routines.

8.3 crash

Demonstrates how to use the optional GBDK crash handler which dumps debug info to the Game Boy screen in the event of a program crash.

8.4 colorbar

The colorbar program, written by Mr. N.U. of TeamKNOx, illustrates the use of colors on a Color GameBoy.

8.5 dscan

Deep Scan is a game written by Mr. N.U. of TeamKNOx that supports the Color GameBoy. Your aim is to destroy the submarines from your boat, and to avoid the projectiles that they send to you. The game should be self-explanatory. The following keys are used:

```
RIGHT/LEFT   : Move your boat
A/B          : Send a bomb from one side of your boat
START        : Start game or pause game

When game is paused:

SELECT       : Invert A and B buttons
RIGHT/LEFT   : Change speed
UP/DOWN      : Change level
```

8.6 filltest

Demonstrates various graphics routines.

8.7 fonts

Examples of how to work with the built in font and printing features.

8.8 galaxy

A C translation of the space.s assembly program.

8.9 gb-dtmf

The gb-dtmf, written by Osamu Ohashi, is a Dual Tone Multi-Frequency (DTMF) generator.

8.10 irq

Illustrates how to install interrupt handlers.

8.11 large map

Shows how to scroll with maps larger than 32 x 32 tiles using set_bkg_submap(). It fills rows and columns at the edges of the visible viewport (of the hardware Background Map) as it scrolls from the desired sub-region of the large map.

8.12 lcd isr wobble

An example of how to use the LCD ISR for visual special effects

8.13 paint

The paint example is a painting program. It supports different painting tools, drawing modes, and colors. At the moment, it only paints individual pixels. This program illustrates the use of the full-screen drawing library. It also illustrates the use of generic structures and big sprites.

```
Arrow keys : Move the cursor
SELECT     : Display/hide the tools palette
A          : Select tool
```

8.14 rand

The rand program, written by Luc Van den Borre, illustrates the use of the GBDK random generator.

8.15 ram_fn

The ram_fn example illustrates how to copy functions to RAM or HIRAM, and how to call them from C.

8.16 rpn

A basic RPN calculator. Try entering expressions like 12 134∗ and then 1789+.

8.17 samptest

Demonstration of playing a sound sample.

8.18 sgb (various)

A collection of examples showing how to use the Super Game Boy API features.

8.19 sound

The sound example is meant for experimenting with the soung generator of the GameBoy (to use on a real Game↩ Boy). The four different sound modes of the GameBoy are available. It also demonstrates the use of bit fields in C (it's a quick hack, so don't expect too much from the code). The following keys are used:

```
UP/DOWN      : Move the cursor
RIGHT/LEFT   : Increment/decrement the value
RIGHT/LEFT+A : Increment/decrement the value by 10
RIGHT/LEFT+B : Set the value to maximum/minimum
START        : Play the current mode's sound (or all modes if in control screen)
START+A      : Play a little music with the current mode's sound
SELECT       : Change the sound mode (1, 2, 3, 4 and control)
SELECT+A     : Dump the sound registers to the screen
```

8.20 space

The space example is an assembly program that demonstrates the use of sprites, window, background, fixed-point values and more. The following keys are used:

```
Arrow keys     : Change the speed (and direction) of the sprite
Arrow keys + A : Change the speed (and direction) of the window
Arrow keys + B : Change the speed (and direction) of the background
START          : Open/close the door
SELECT         : Basic fading effect
```

8.21 templates

Two basic template examples are provided as a starting place for writing your GBDK programs.

9 Frequently Asked Questions (FAQ)

9.1 Frequently Asked Questions

- How do I set the ROM's title?
  - Use the makebin `-yn` flag. For example with lcc `-Wm-yn"MYTITLE"` or with makebin directly `-yn "MYTITLE"`. The maximum length is up to 15 characters, but may be shorter.
  - See "0134-0143 - Title" in Pandocs for more details.

- How do I set SGB, Color only and Color compatibility in the ROM header?
  - Use the following makebin flags. Prefix them with `-Wm` if using lcc.
    - `-yc` : GameBoy Color compatible
    - `-yC` : GameBoy Color only
    - `-ys` : Super GameBoy compatible

- How do I set the ROM MBC type?
  - See setting_mbc_and_rom_ram_banks

- What do these kinds of warnings / errors mean? `WARNING: possibly wrote twice at addr 4000 (93->3E)  Warning:  Write from one bank spans into the next.  7ff7 -> 8016 (bank 1 -> 2)`

  You may have a overflow in one of your ROM banks. If there is more data allocated to a bank than it can hold it then will spill over into the next bank. The warnings are generated by ihxcheck during conversion of an .ihx file into a ROM file.

  See the section ROM/RAM Banking and MBCs for more details about how banks work and what their size is. You may want to use a tool such as romusage to calculate the amount of free and used space.

- Why is the compiler so slow, or why did it suddenly get much slower?
  - This may happen if you have large initialized arrays declared without the `const` keyword. It's important to use the const keyword for read-only data. See const_gbtd_gbmb and const_array_data

- What flags should be enabled for debugging?
  - You can use the lcc debug flag

- Why are 8 bit numbers not printing correctly with printf()?
  - To correctly pass chars/uint8s for printing, they must be explicitly re-cast as such when calling the function. See docs_chars_varargs for more details.

- How can maps larger than 32x32 tiles be scrolled? & Why is the map wrapping around to the left side when setting a map wider than 32 tiles with set_bkg_data()?
  - The hardware Background map is 32 x 32 tiles. The screen viewport that can be scrolled around that map is 20 x 18 tiles. In order to scroll around within a much larger map, new tiles must be loaded at the edges of the screen viewport in the direction that it is being scrolled. set_bkg_submap can be used to load those rows and columns of tiles from the desired sub-region of the large map.
  - See the "Large Map" example program and set_bkg_submap()
  - Writes that exceed coordinate 31 of the Background tile map on the x or y axis will wrap around to the Left and Top edges.

- When using gbt_player with music in banks, how can the current bank be restored after calling gbt_update()? (since it changes the currently active bank without restoring it).
  - See restoring the current bank

10 Migrating to new GBDK Versions

This section contains information that may be useful to know or important when upgrading to a newer GBDK release.

10.1 GBDK 2020 versions

10.1.1 Porting to GBDK 2020 4.0.3

- No significant changes required

10.1.2 Porting to GBDK 2020 4.0.2

- The default font has been reduced from 256 to 96 characters.
  - Code using special characters may need to be updated.
  - The off-by-1 character index offset was removed for fonts. Old fonts with the offset need to be re-adjusted.

10.1.3 Porting to GBDK 2020 4.0.1

- **Important!** : The `WRAM` memory region is no longer automatically initialized to zeros during startup.
  - Any variables which are declared without being initialized may have **indeterminate values instead of 0** on startup. This might reveal previously hidden bugs in your code.
  - Check your code for variables that are not initialized before use.
  - In BGB you can turn on triggering exceptions (options panel) reading from unitialized RAM. This allows for some additional runtime detection of uninitialized vars.
- In .ihx files, multiple writes to the same ROM address are now warned about using ihxcheck.
- `set_*_tiles()` now wrap maps around horizontal and vertical boundaries correctly. Code relying on it not wrapping correctly may be affected.

10.1.4 Porting to GBDK 2020 4.0

- GBDK now requires SDCC 4.0.3 or higher
- The old linker `link-gbz80` has been REMOVED, the linker sdldgb from SDCC is used.
  - Due to the linker change, there are no longer warnings about multiple writes to the same ROM address.
- GBDK now generates .ihx files, those are converted to a ROM using makebin (lcc can do this automatically in some use cases)
- Setting ROM bytes directly with `-Wl-yp0x<address>=0x<value>` is no longer supported. Instead use makebin flags. For example, use `-Wm-yC` instead of `-Wl-yp0x143=0xC0`.
- OAM symbol has been renamed to _shadow_OAM, that allows accessing shadow OAM directly from C code

10.1.5 Porting to GBDK 2020 3.2

- No significant changes required

10.1.6 Porting to GBDK 2020 3.1.1

- No significant changes required

10.1.7 Porting to GBDK 2020 3.1

- No significant changes required

10.1.8 Porting to GBDK 2020 3.0.1

- LCC was upgraded to use SDCC v4.0. Makefile changes may be required
  - The symbol format changed. To get usable symbols turn on `.noi` output (LCC argument: `-Wl-j`)can be enabled and you can use noi2sym
  - ?? Suggested: With LCC argument: `-Wa-l` (sdasgb:`-a All user symbols made global`)
  - In SDCC 3.6.0, the default for char changed from signed to unsigned.
    - If you want the old behavior use `--fsigned-char`.
    - lcc includes `--fsigned-char` by default
    - Explicit declaration of unsigned vars is encouraged (for example, '15U' instead of '15')
  - `.init` address has been removed

10.2 Historical GBDK versions

10.2.1 GBDK 1.1 to GBDK 2.0

- Change your int variables to long if they have to be bigger than 255. If they should only contain values between 0 and 255, use an unsigned int.
- If your application uses the delay function, you'll have to adapt your delay values.
- Several functions have new names. In particular some of them have been changed to macros (e.g. show_↩ bkg() is now SHOW_BKG).
- You will probably have to change the name of the header files that you include.

11 GBDK Releases

The GBDK 2020 releases can be found on Github: https://github.com/Zal0/gbdk-2020/releases

11.1 GBDK 2020 Release Notes

11.1.1 GBDK 2020 4.0.3

2021/03

- Library
  - Added set_vram_byte()
  - Added set_bkg_tile_xy() / set_win_tile_xy()
  - Added get_bkg_xy_addr() / get_win_xy_addr()
  - Added get_bkg_submap() / get_win_submap()
  - Added metasprite api support
  - Added gb_decompress support
  - Added calloc / malloc / realloc / free and generic memmove
  - Improved printf(): ignore %0 padding and %1-9 width specifier instead of not printing, support upper case X
  - Fixed line(): handle drawing when x1 is less than x2
- Examples
  - Added large_map: showing how to use set_bkg_submap()
  - Added scroller: showing use of get_bkg_xy_addr(), set_bkg_tile_xy() and set_vram_byte
  - Added gbdecompress: de-compressing tile data into vram
  - Added template projects
  - Fixed build issue with banks_autobank example
  - Improved sgb_border
- Added GBCompress utility
- Added png2metaspr utility and metasprites example

11.1.2 GBDK 2020 4.0.2

2021/01/17

- Includes SDCC snapshot build version 12016 (has a fix for duplicate debug symbols generated from inlined header functions which GBDK 4.0+ uses)
- Updated documentation

- Library was improved
  - Linking with stdio.h does not require that much ROM now
  - Default font is changed to the smaller one (102 characters), that leaves space for user tiles
  - Fixed broken support for multiplying longs
  - memset/memcpy minor enhancements
  - safer copy-to-VRAM functions
  - loading of 1bit data fixed, also now it is possible to specify pixel color
  - Improved code generation for the GBDK Library with SDCC switch on by default: `--max-allocs-per-node 50000`
  - fixed wrong parameter offsets in hiramcpy() (broken ram_function example)
  - Multiple minor improvements
- New bankpack feature, allows automatic bank allocation for data and code, see banks_autobank example, feature is in beta state, use with care
- Lcc improvements
  - Fixed option to specify alternate base addresses for shadow_OAM, etc
- Examples: Added bgb debug example

11.1.3 GBDK 2020 4.0.1

2020/11/14

- Updated API documentation
- IHX is checked for correctness before the makebin stage. That allows to warn about overwriting the same ROM addresses (SDCC toolchain does not check this anymore).
- Library was improved
  - set_*_tiles() now wrap maps around horizontal and vertical boundaries correctly
  - new fill_*_rect() functions to clear rectangle areas
  - runtime initialization code now does not initialize whole WRAM with zeros anymore, that allows BGB to raise exceptions when code tries to read WRAM that was not written before.
  - enhanced SGB support
    - joypad_init() / joypad_ex() support for multiple joypads
    - SGB border example
  - _current_bank variable is updated when using bank switching macros
  - Reorganized examples: each example is in separate folder now, that simplifies understanding.
  - Lcc improvements
    - Fix -S flag
    - Fix default stack location from 0xDEFF to 0xE000 (end of WRAM1)
    - Fix cleanup of .adb files with -Wf–debug flag
    - Fix output not working if target is -o some_filename.ihx

11.1.4 GBDK 2020 4.0

2020/10/01

- GBDK now requires SDCC 4.0.3 or higher, that has fully working toolchain. Old link-gbz80 linker is not used anymore, sdldgb and makebin are used to link objects and produce binary roms; maccer tool is no longer needed either
  - SDCC 4.0.3 has much better code generator which produces smaller and faster code. Code is twice faster

  - SOURCE LEVEL DEBUGGING is possible now! Native toolchain produces ∗.CDB files that contain detailed debug info. Look for EMULICIOUS extension for vs.code. It supports breakpoints, watches, inspection of local variables, and more!
  - SDCC 4.0.4 has fixed RGBDS support; library is not updated to support that in full yet, but it is possible to assemble and link code emitted by SDCC with RGDBS
  - New banked trampolines are used, they are faster and smaller
  - New (old) initialization for non-constant arrays do NOT require 5 times larger rom space than initialized array itself, SDCC even tries to compress the data

- Library was improved
  - itoa/ltoa functions were rewritten, div/mod is not required now which is about 10 times faster
  - sprite functions are inline now, which is faster up to 12 times and produces the same or smaller code; .OAM symbol is renamed into _shadow_OAM that allows accessing shadow OAM directly from C code
  - interrupt handling was revised, it is now possible to make dedicated ISR's, that is important for time-sensitive handlers such as HBlank.
  - printf/sprintf were rewritten and splitted, print functions are twice faster now and also requre less rom space if you use sprintf() only, say, in bgb_emu.h
  - crash_handler.h - crash handler that allows to detect problems with ROMs after they are being released (adapted handler, originally written by ISSOtm)
  - improved and fixed string.h
  - many other improvements and fixes - thanks to all contributors!
- Revised examples
- Improved linux support
- Lcc has been updated
  - it works with the latest version of sdcc
  - quoted paths with spaces are working now

11.1.5 GBDK 2020 3.2

2020/06/05

- Fixed OAM initialization that was causing a bad access to VRAM
- Interrupt handlers now wait for lcd controller mode 0 or 1 by default to prevent access to inaccessible VRAM in several functions (like set_bkg_tiles)
- Several optimizations here and there

11.1.6 GBDK 2020 3.1.1

2020/05/17

- Fixed issues with libgcc_s_dw2-1.dll

11.1.7 GBDK 2020 3.1

2020/05/16

- Banked functions are working! The patcher is fully integrated in link-gbz80, no extra tools are needed. It is based on Toxa's work
  - Check this post for more info
  - Check the examples/gb/banked code for basic usage

- USE_SFR_FOR_REG is the default now check here why https://gbdev.gg8.se/forums/viewtopic.↩ php?id=697
- Fixed examples that were not compiling in the previous version and some improvements in a few of them. Removed all warnings caused by changing to the new SDCC
- Fixed bug in lcc that was causing some files in the temp folder not being deleted
- Removed as-gbz80 (the lib is now compiled with sdasgb thanks to this workaround) https↩ ://github.com/Zal0/gbdk-2020/commit/d2caafa4a66eb08998a14b258cb66af041a0e5c8
- Profile support with bgb emulator
  - Basic support including <gb/bgb_emu.h> and using the macros BGB_PROFILE_BEGIN and BG↩ B_PROFILE_END. More info in this post https://gbdev.gg8.se/forums/viewtopic.↩ php?id=703
  - For full profiling check this repo and this post https://github.com/untoxa/bgb_↩ profiling_toolkit/blob/master/readme.md https://gbdev.gg8.se/forums/viewtopic.↩ php?id=710

11.1.8 GBDK 2020 3.0.1

2020/04/12

- Updated SDCC to v.4.0
- Updated LCC to work with the new compiler

11.1.9 GBDK 2020 3.0

2020/04/12

- Initial GBDK 2020 release
  Updated SDCC to v4.0 The new linker is not working so the old version is still there There is an issue with sdagb compiling drawing.s (the JP in line 32 after ".org .MODE_TABLE+4∗.G_MODE" it's writing more than 4 bytes invading some addresses required by input.s:41) Because of this, all .s files in libc have been assembled with the old as-gbz80 and that's why it is still included

11.2 Historical GBDK Release Notes

11.2.1 GBDK 2.96

17 April, 2000
Many changes.

- Code generated is now much more reliable and passes all of sdcc's regression suite.
- Added support for large sets of local variables (>127 bytes).
- Added full 32 bit long support.
- Still no floating pt support.

11.2.2 GBDK 2.95-3

19th August, 2000

- Stopped lcc with sdcc from leaking .cdb files all across /tmp.
- Optimised < and > for 16 bit varibles.

- Added a new lexer to sdcc. Compiling files with large initalised arrays takes 31% of the time (well, at least samptest.c does :)

This is an experimental release for those who feel keen. The main change is a new lexer (the first part in the compilation process which recognises words and symbols like '!=' and 'char' and turns them into a token number) which speeds up compilation of large initialised arrays like tile data by a factor of three. Please report any bugs that show up - this is a big change.
I have also included a 'minimal' release for win32 users which omits the documentation, library sources, and examples. If this is useful I will keep doing it.

11.2.3 GBDK 2.95-2

5th August, 2000
Just a small update. From the README:

- Added model switching support –model-medium uses near (16 bit) pointers for data, and banked calls for anything not declared as 'nonbanked' –model-small uses near (16 bit) pointers for data and calls. Nothing uses banked calls. 'nonbanked' functions are still placed in HOME. Libraries are under lib/medium and lib/small.
- Added the gbdk version to 'sdcc –version'
- Changed the ways globals are exported, reducing the amount of extra junk linked in.
- Turned on the optimisations in flex. Large constant arrays like tile data should compile a bit faster.

11.2.4 GBDK 2.95

22nd July, 2000

- Fixed 'a $<<$ c' for c = [9..15]
- no$gmb doesn't support labels of $>$ 32 chars. The linker now trims all labels to 31 chars long.
- Fixed wait_vbl for the case where you miss a vbl
- Fixed + and - for any type where sizeof == 2 and one of the terms was on the stack. This includes pointers and ints. Fixes the text output bug in the examples. Should be faster now as well. Note that + and - for longs is still broken.
- Fixed the missing $*$/ in gb.h
- Added basic far function support. Currently only works for isas and rgbasm. See examples/gb/far/$*$
- bc is now only pushed if the function uses it. i.e. something like: int silly(int i) { return i; }
  will not have the push bc; pop bc around it.
- Better rgbasm support. Basically: o Use "sdcc -mgbz80 --asm=rgbds file.c" for each file.c o Use "sdcc -mgbz80 --asm=rgbds crt0.o gbz80.lib gb.lib file1.o file2.o..."

to link everything together. The .lib files are generated using astorgb.pl and sdcc to turn the gbdk libraries into something rgbds compatible. The libraries are *not* fully tested. Trust nothing. But give it a go :)

- Ran a spell checker across the README and ChangeLog

This is a recommended upgrade. Some of the big features are:
Decent rgbds support. All the libraries and most of the examples can now compile with rgbds as the assembler. Banked function support. It is now easier to break the 32k barrier from within C. Functions can live in and be called transparently from any bank. Only works with rgbds Fixed some decent bugs with RSH, LSH, and a nasty bug with + and - for int's and pointers. Various optimisations in the code generator.
7th July, 2000
Information on float and long support. Someone asked about the state of float/long support recently. Heres my reply:

long support is partly there, as is float support. The compiler will correctly recognise the long and float keywords, and will generate the code for most basic ops (+, -, &, | etc) for longs correctly and will generate the function calls for floats and hard long operations (*, /, %) correctly. However it wont generate float constants in the correct format, nor will it 'return' a long or float - gbdk doesn't yet support returning types of 4 bytes. Unfortunately its not going to make it into 2.95 as there's too much else to do, but I should be able to complete long support for 2.96

11.2.5 GBDK 2.94

7th May, 2000
Many fixes - see the README for more.
7th May - Library documentation up. A good size part of the libraries that go with gbdk have been documented - follow the HTML link above to have a look. Thanks to quang for a good chunk of the gb.h documentation. Please report any errors :)

- Fixed #define BLAH 7 // Unterminated ' error in sdcpp
  - Fixed SCY_REG += 2, SCY_REG -= 5 (add and subtract in indirect space) as they were both quite broken.
  - externs and static's now work as expected.
  - You can now specify which bank code should be put into using a #pragma e.g: #pragma bank=HOME Under rgbds and asxxxx putting code in the HOME bank will force the code into bank 0 - useful for library functions. The most recent #pragma bank= will be the one used for the whole file.
  - Fixed an interesting bug in the caching of lit addresses
  - Added support for accessing high registers directly using the 'sfr' directive. See libc/gb/sfr.s and gb/hardware.h for an example. It should be possible with a bit of work to make high ram directly usable by the compiler; at the moment it is experimental. You can test sfr's by enabling USE_SFR_FOR_R↩ EG=1
  - Added remove_VBL etc functions.
  - Documented the libs - see the gbdk-doc tarball distributed seperatly.
  - Two dimensional arrays seem to be broken.

11.2.6 GBDK 2.93

6th April, 2000
From the README

- Added multi-bank support into the compiler - The old -Wf-boxx and -Wf-baxx options now work
- Has preliminary support for generating rgbds and ISAS compatible assembler. Try -W–asm=rgbds or -W–asm=isas. The ISAS code is untested as I dont have access to the real assembler.
- RSH is fixed
- AND is fixed
- The missing parts of 2.1.0's libs are there. Note: They are untested.
- The dscan demo now fully works (with a hack :)
- There is a bug with cached computed values which are later used as pointers. When the value is first used as a BYTE arg, then later as a pointer the pointer fails as the high byte was never computed and is now missing. A temporary fix is to declare something appropriate as 'volatile' to stop the value being cached. See dscan.c/bombs() for an example.

11.2.7 GBDK 2.92-2 for win32

26th March, 2000
This is a maintenance release for win32 which fixes some of the niggly install problems, especially:

- win32 only. Takes care of some of the install bugs, including:
  - Now auto detects where it is installed. This can be overridden using set GBDKDIR=...
  - Problems with the installer (now uses WinZip)
  - Problems with the temp directory Now scans TMP, TEMP, TMPDIR and finally c: tmp
  - cygwin1.dll and 'make' are no longer required gbdk is now built using mingw32 which is win32 native make.bat is automagically generated from the Makefile
  - I've reverted to using WORD for signed 16 bit etc. GBDK_2_COMPAT is no longer required.

WORDS are now back to signed. GBDK_2_COMPAT is no longer needed. Temporary files are created in T↩ MP, TEMP, or TMPDIR instead of c: tmp The installer is no more as it's not needed. There is a WinZip wrapped version for those with the extra bandwidth :). gbdk autodetects where it is installed - no more environment variables. cygwin1.dll and make are no longer required - gbdk is now compiled with mingw32.
See the ChangeLog section in the README for more information.
21st March, 2000
Problems with the installer. It seems that the demo of InstallVISE has an unreasonably short time limit. I had planed to use the demo until the license key came through, but there's no sign of the key yet and the 3 day evaluation is up. If anyone knows of a free Windows installer with the ability to modify environment variables, please contact me. I hear that temporarily setting you clock back to the 15th works...
18th March, 2000
libc5 version available / "Error creating temp file" Thanks to Rodrigo Couto there is now a Linux/libc5 version of gbdk3-2.92 available - follow the download link above. At least it will be there when the main sourceforge site comes back up... Also some people have reported a bug where the compiler reports '∗∗∗ Error creating temp file'. Try typing "mkdir c: tmp" from a DOS prompt and see if that helps.

11.2.8 GBDK 2.92

8th March, 2000
Better than 2.91 :). Can now be installed anywhere. All the demos work. See the README for more.

- All the examples now work (with a little bit of patching :)
  - Fixed problem with registers being cached instead of being marked volatile.
  - More register packing - should be a bit faster.
  - You can now install somewhere except c: gbdk | /usr/lib/gbdk
  - Arrays initialised with constant addresses a'la galaxy.c now work.
  - Fixed minor bug with 104$: labels in as.
  - Up to 167d/s...

11.2.9 GBDK 2.91

27th Feb, 2000
Better than 2.90 and includes Linux, win32 and a source tar ball. Some notes:
Read the README first Linux users need libgc-4 or above. Debian users try apt-get install libgc5. All the types have changed. Again, please read the README first. I prefer release early, release often. The idea is to get the bugs out there so that they can be squashed quickly. I've split up the libs so that they can be used on other platforms and so that the libs can be updated without updating the compiler. One side effect is that gb specific files have been shifted into their own directory i.e. gb.h is now gb/gb.h.
23rd Feb, 2000
First release of gbdk/sdcc. This is an early release - the only binary is for Linux and the source is only available through cvs. If your interested in the source, have a look at the cvs repository gbdk-support first, which will download all the rest of the code. Alternatively, look at gbdk-support and gbdk-lib at cvs.gbdk.sourceforge.net and sdcc at

cvs.sdcc.sourceforge.net. I will be working on binaries for Win32 and a source tar ball soon. Please report any bugs through the bugs link above.
31st Jan, 2000
Added Dermot's far pointer spec. It's mainly here for comment. If sdcc is ported to the Gameboy then I will be looking for some way to do far calls.
8th Jan, 2000
Moved over to sourceforge.net. Thanks must go to David Pfeffer for gbdk's previous resting place, www.gbdev.org. The transition is not complete, but cvs and web have been shifted. Note that the cvs download instructions are stale - you should now look to cvs.gbdk.sourceforge.net. I am currently working on porting sdcc over to the Z80. David Nathan is looking at porting it to the GB.
6th Jan, 2000
Icehawk wrote "I did write some rumble pack routines. Just make sure to remind people to add -Wl-yt0x1C or -Wl-yt0x1D or -Wl-yt0x1E depending on sram and battery usage. Find the routines on my site (as usual). =)"
18th Oct, 1999
Bug tracking / FAQ up. Try the link on the left to report any bugs with GBDK. It's also the first place to look if your having problems.

11.2.10 GBDK 2.1.5

17th Oct, 1999

The compiler is the same, but some of the libraries have been improved. memset() and memcpy() are much faster, malloc() is fixed, and a high speed fixed block alternative malloc() was added.

12 Todo List

Page Coding Guidelines

Update and verify this section for the modernized SDCC and toolchain

File far_ptr.h

Add link to a discussion about banking (such as, how to assign code and variables to banks)

Page GBDK Toolchain

Document png2mtspr

File malloc.h

: This library may currently be broken.

Page ROM/RAM Banking and MBCs

Fill in this info for Banked Functions Banked functions (located in a switchable ROM bank)

- May call functions in any bank: ?
- May use data in any bank: **NO** (may only use data from currently active banks)

Const Data (Variables in ROM)

Variables in RAM

Page Using GBDK

This is from GBDK 2.x docs, verify it with GBDK-2020 and modern SDCC

13 Module Index

13.1 C modules

Here is a list of all modules:

14 Data Structure Index

14.1 Data Structures

Here are the data structures with brief descriptions:

15 File Index

15.1 File List

Here is a list of all files with brief descriptions:

16 Module Documentation

16.1 List of gbdk fonts

16.1.1 Description

Variables

- UINT8 font_spect [ ]
- UINT8 font_italic [ ]
- UINT8 font_ibm [ ]
- UINT8 font_min [ ]
- UINT8 font_ibm_fixed [ ]

16.1.2 Variable Documentation

16.1.2.1 font_spect `UINT8 font_spect[ ]`

The default fonts

16.1.2.2 font_italic UINT8 font_italic[]

16.1.2.3 font_ibm UINT8 font_ibm[]

16.1.2.4 font_min UINT8 font_min[]

16.1.2.5 font_ibm_fixed UINT8 font_ibm_fixed[]

Backwards compatible font

17 Data Structure Documentation

17.1 __far_ptr Union Reference

`#include <far_ptr.h>`

Data Fields

- FAR_PTR ptr
- struct {
  void * ofs
  unsigned int seg
  } segofs

- struct {
  void(* fn)()
  unsigned int seg
  } segfn

17.1.1 Detailed Description

Union for working with members of a FAR_PTR

17.1.2 Field Documentation

17.1.2.1 ptr FAR_PTR __far_ptr::ptr

17.1.2.2 ofs void* __far_ptr::ofs

17.1.2.3 seg unsigned int __far_ptr::seg

17.1.2.4 segofs struct { ... } __far_ptr::segofs

17.1.2.5 fn void(* __far_ptr::fn) ()

17.1.2.6 segfn `struct { ... } __far_ptr::segfn`

The documentation for this union was generated from the following file:

- gb/far_ptr.h

17.2 _fixed Union Reference

`#include <types.h>`

Data Fields

- struct {
  UBYTE l
  UBYTE h
  } b
- UWORD w

17.2.1 Detailed Description

Useful definition for fixed point values

17.2.2 Field Documentation

17.2.2.1 l `UBYTE _fixed::l`

17.2.2.2 h `UBYTE _fixed::h`

17.2.2.3 b `struct { ... } _fixed::b`

17.2.2.4 w `UWORD _fixed::w`

The documentation for this union was generated from the following file:

- asm/types.h

17.3 atomic_flag Struct Reference

`#include <stdatomic.h>`

Data Fields

- unsigned char flag

17.3.1 Field Documentation

17.3.1.1 flag `unsigned char atomic_flag::flag`

The documentation for this struct was generated from the following file:

- stdatomic.h

17.4 joypads_t Struct Reference

`#include <gb.h>`

Data Fields

- UINT8 npads
- union {
  struct {
  UINT8 joy0
  UINT8 joy1
  UINT8 joy2
  UINT8 joy3
  }
  UINT8 joypads [4]
  };

17.4.1 Detailed Description

Multiplayer joypad structure.
Must be initialized with joypad_init() first then it may be used to poll all avaliable joypads with joypad_ex()

17.4.2 Field Documentation

17.4.2.1 npads `UINT8 joypads_t::npads`

17.4.2.2 joy0 `UINT8 joypads_t::joy0`

17.4.2.3 joy1 `UINT8 joypads_t::joy1`

17.4.2.4 joy2 `UINT8 joypads_t::joy2`

17.4.2.5 joy3 `UINT8 joypads_t::joy3`

17.4.2.6 joypads `UINT8 joypads_t::joypads[4]`

17.4.2.7 "@4 `union { ... }`

The documentation for this struct was generated from the following file:

- gb/gb.h

17.5 metasprite_t Struct Reference

`#include <metasprites.h>`

Data Fields

- INT8 dy
- INT8 dx
- UINT8 dtile
- UINT8 props

17.5.1 Detailed Description

metasprite item description

17.5.2 Field Documentation

17.5.2.1 dy `INT8 metasprite_t::dy`

17.5.2.2 dx `INT8 metasprite_t::dx`

17.5.2.3 dtile `UINT8 metasprite_t::dtile`

17.5.2.4 props `UINT8 metasprite_t::props`

The documentation for this struct was generated from the following file:

- gb/metasprites.h

17.6 OAM_item_t Struct Reference

`#include <gb.h>`

Data Fields

- UINT8 y
- UINT8 x
- UINT8 tile
- UINT8 prop

17.6.1 Detailed Description

Sprite Attributes structure

Parameters

x	X Coordinate of the sprite on screen
y	Y Coordinate of the sprite on screen
tile	Sprite tile number (see set_sprite_tile)
prop	OAM Property Flags (see set_sprite_prop)

17.6.2 Field Documentation

17.6.2.1 y `UINT8 OAM_item_t::y`

17.6.2.2 x UINT8 OAM_item_t::x

17.6.2.3 tile UINT8 OAM_item_t::tile

17.6.2.4 prop UINT8 OAM_item_t::prop

The documentation for this struct was generated from the following file:

- gb/gb.h

17.7 sfont_handle Struct Reference

`#include <font.h>`

Data Fields

- UINT8 first_tile
- void * font

17.7.1 Detailed Description

Font handle structure

17.7.2 Field Documentation

17.7.2.1 first_tile UINT8 sfont_handle::first_tile

First tile used for font

17.7.2.2 font void* sfont_handle::font

Pointer to the base of the font

The documentation for this struct was generated from the following file:

- gb/font.h

17.8 smalloc_hunk Struct Reference

`#include <malloc.h>`

Data Fields

- UBYTE magic
- pmmalloc_hunk next
- UWORD size
- int status

17.8.1 Field Documentation

17.8.1.1 magic UBYTE smalloc_hunk::magic

17.8.1.2 next `pmmalloc_hunk smalloc_hunk::next`

17.8.1.3 size `UWORD smalloc_hunk::size`

17.8.1.4 status `int smalloc_hunk::status`

The documentation for this struct was generated from the following file:

- gb/malloc.h

18 File Documentation

18.1 /home/birch/git/gbdev/gbdk2020/gbdk-2020-git/docs/pages/01_getting_started.md File Reference

18.2 /home/birch/git/gbdev/gbdk2020/gbdk-2020-git/docs/pages/02_links_and_tools.md File Reference

18.3 /home/birch/git/gbdev/gbdk2020/gbdk-2020-git/docs/pages/03_using_gbdk.md File Reference

18.4 /home/birch/git/gbdev/gbdk2020/gbdk-2020-git/docs/pages/04_coding_guidelines.md File Reference

18.5 /home/birch/git/gbdev/gbdk2020/gbdk-2020-git/docs/pages/05_banking_mbcs.md File Reference

18.6 /home/birch/git/gbdev/gbdk2020/gbdk-2020-git/docs/pages/06_toolchain.md File Reference

18.7 /home/birch/git/gbdev/gbdk2020/gbdk-2020-git/docs/pages/07_sample_programs.md File Reference

18.8 /home/birch/git/gbdev/gbdk2020/gbdk-2020-git/docs/pages/08_faq.md File Reference

18.9 /home/birch/git/gbdev/gbdk2020/gbdk-2020-git/docs/pages/09_migrating_new_versions.md File Reference

18.10 /home/birch/git/gbdev/gbdk2020/gbdk-2020-git/docs/pages/10_release_notes.md File Reference

18.11 /home/birch/git/gbdev/gbdk2020/gbdk-2020-git/docs/pages/docs_index.md File Reference

18.12 asm/gbz80/provides.h File Reference

Macros

- #define USE_C_MEMCPY 0
- #define USE_C_STRCPY 0
- #define USE_C_STRCMP 0

18.12.1 Macro Definition Documentation

18.12.1.1 USE_C_MEMCPY `#define USE_C_MEMCPY 0`

18.12.1.2 USE_C_STRCPY `#define USE_C_STRCPY 0`

18.12.1.3 USE_C_STRCMP `#define USE_C_STRCMP 0`

18.13 asm/gbz80/stdarg.h File Reference

Macros

- #define va_start(list, last) list = (unsigned char *)&last + sizeof(last)
- #define va_arg(list, type) *((type *)((list += sizeof(type)) - sizeof(type)))
- #define va_end(list)

Typedefs

- typedef unsigned char * va_list

18.13.1 Macro Definition Documentation

18.13.1.1 va_start

```
#define va_start(
    list,
    last ) list = (unsigned char *)&last + sizeof(last)
```

18.13.1.2 va_arg

```
#define va_arg(
    list,
    type ) *((type *)((list += sizeof(type)) - sizeof(type)))
```

18.13.1.3 va_end

```
#define va_end(
    list )
```

18.13.2 Typedef Documentation

18.13.2.1 va_list `typedef unsigned char* va_list`

18.14 stdarg.h File Reference

`#include <asm/gbz80/stdarg.h>`

18.15 asm/gbz80/types.h File Reference

Macros

- #define NONBANKED __nonbanked
- #define BANKED __banked
- #define CRITICAL __critical
- #define INTERRUPT __interrupt
- #define __SIZE_T_DEFINED

Typedefs

- typedef signed char INT8
- typedef unsigned char UINT8
- typedef signed int INT16
- typedef unsigned int UINT16
- typedef signed long INT32
- typedef unsigned long UINT32
- typedef int size_t
- typedef UINT16 clock_t

18.15.1 Detailed Description

Types definitions for the gb.

18.15.2 Macro Definition Documentation

18.15.2.1 NONBANKED `#define NONBANKED __nonbanked`

18.15.2.2 BANKED `#define BANKED __banked`

18.15.2.3 CRITICAL `#define CRITICAL __critical`

18.15.2.4 INTERRUPT `#define INTERRUPT __interrupt`

18.15.2.5 __SIZE_T_DEFINED `#define __SIZE_T_DEFINED`

18.15.3 Typedef Documentation

18.15.3.1 INT8 `typedef signed char INT8`

Signed eight bit.

18.15.3.2 UINT8 `typedef unsigned char UINT8`

Unsigned eight bit.

18.15.3.3 INT16 `typedef signed int INT16`

Signed sixteen bit.

18.15.3.4 UINT16 typedef unsigned int UINT16

Unsigned sixteen bit.

18.15.3.5 INT32 typedef signed long INT32

Signed 32 bit.

18.15.3.6 UINT32 typedef unsigned long UINT32

Unsigned 32 bit.

18.15.3.7 size_t typedef int size_t

18.15.3.8 clock_t typedef UINT16 clock_t

Returned from clock

See also

clock

18.16 asm/types.h File Reference

```
#include <asm/gbz80/types.h>
```

Data Structures

- union _fixed

Typedefs

- typedef INT8 BOOLEAN
- typedef INT8 BYTE
- typedef UINT8 UBYTE
- typedef INT16 WORD
- typedef UINT16 UWORD
- typedef INT32 LWORD
- typedef UINT32 ULWORD
- typedef INT32 DWORD
- typedef UINT32 UDWORD
- typedef union _fixed fixed

18.16.1 Detailed Description

Shared types definitions.

18.16.2 Typedef Documentation

18.16.2.1 BOOLEAN typedef INT8 BOOLEAN

TRUE or FALSE.

18.16.2.2 BYTE typedef INT8 BYTE

Signed 8 bit.

18.16.2.3 UBYTE typedef UINT8 UBYTE

Unsigned 8 bit.

18.16.2.4 WORD typedef INT16 WORD

Signed 16 bit

18.16.2.5 UWORD typedef UINT16 UWORD

Unsigned 16 bit

18.16.2.6 LWORD typedef INT32 LWORD

Signed 32 bit

18.16.2.7 ULWORD typedef UINT32 ULWORD

Unsigned 32 bit

18.16.2.8 DWORD typedef INT32 DWORD

Signed 32 bit

18.16.2.9 UDWORD typedef UINT32 UDWORD

Unsigned 32 bit

18.16.2.10 fixed typedef union _fixed fixed

Useful definition for fixed point values

18.17 types.h File Reference

```
#include <asm/types.h>
```

Macros

- #define NULL 0
- #define FALSE 0
- #define TRUE 1

Typedefs

- typedef void * POINTER

18.17.1 Detailed Description

Basic types.
Directly include the port specific file.

18.17.2 Macro Definition Documentation

18.17.2.1 NULL #define NULL 0

Good 'ol NULL.

18.17.2.2 FALSE #define FALSE 0

A 'false' value.

18.17.2.3 TRUE #define TRUE 1

A 'true' value.

18.17.3 Typedef Documentation

18.17.3.1 POINTER `typedef void* POINTER`

No longer used.

18.18 assert.h File Reference

Macros

- #define assert(x) ((x) ? (void)0 : __assert(#x, __func__, __FILE__, __LINE__))

Functions

- void __assert (const char *expression, const char *functionname, const char *filename, unsigned int linenumber)

18.18.1 Macro Definition Documentation

18.18.1.1 assert

```
#define assert(
        x ) ((x) ?  (void)0 :  __assert(#x, __func__, __FILE__, __LINE__))
```

18.18.2 Function Documentation

18.18.2.1 __assert()

```
void __assert (
    const char * expression,
    const char * functionname,
    const char * filename,
    unsigned int linenumber )
```

18.19 bcd.h File Reference

```
#include <asm/types.h>
```

Macros

- #define BCD_HEX(v) ((BCD)(v))
- #define MAKE_BCD(v) BCD_HEX(0x ## v)

Typedefs

- typedef unsigned long BCD

Functions

- void uint2bcd (unsigned int i, BCD *value)
- void bcd_add (BCD *sour, const BCD *value)
- void bcd_sub (BCD *sour, const BCD *value)
- UBYTE bcd2text (const BCD *bcd, UBYTE tile_offset, unsigned char *buffer)

18.19.1 Detailed Description

Support for working with BCD (Binary Coded Decimal)
See the example BCD project for additional details.

18.19.2 Macro Definition Documentation

18.19.2.1 BCD_HEX `#define BCD_HEX( v ) ((BCD)(v))`

18.19.2.2 MAKE_BCD `#define MAKE_BCD( v ) BCD_HEX(0x ## v)`

Converts an integer value into BCD format
A maximum of 8 digits may be used

18.19.3 Typedef Documentation

18.19.3.1 BCD `typedef unsigned long BCD`

18.19.4 Function Documentation

18.19.4.1 uint2bcd() `void uint2bcd ( unsigned int i, BCD * value )`

Converts integer **i** into BCD format (Binary Coded Decimal)

Parameters

i	Numeric value to convert
value	Pointer to a BCD variable to store the converted result

18.19.4.2 bcd_add() `void bcd_add ( BCD * sour, const BCD * value )`

Adds two numbers in BCD format: **sour** += **value**

Parameters

sour	Pointer to a BCD value to add to (and where the result is stored)
value	Pointer to the BCD value to add to **sour**

18.19.4.3 bcd_sub() `void bcd_sub ( BCD * sour, const BCD * value )`

Subtracts two numbers in BCD format: **sour** -= **value**

Parameters

sour	Pointer to a BCD value to subtract from (and where the result is stored)
value	Pointer to the BCD value to subtract from **sour**

18.19.4.4 bcd2text()

```
UBYTE bcd2text (
    const BCD * bcd,
    UBYTE tile_offset,
    unsigned char * buffer )
```

Convert a BCD number into an asciiz (null terminated) string and return the length

Parameters

bcd	Pointer to BCD value to convert
tile_offset	Optional per-character offset value to add (use 0 for none)
buffer	Buffer to store the result in

Returns: Length in characters (always 8)
buffer should be large enough to store the converted string (9 bytes: 8 characters + 1 for terminator)
There are a couple different ways to use **tile_offset**. For example:

- It can be the Index of the Font Tile '0' in VRAM to allow the buffer to be used directly with set_bkg_tiles.
- It can also be set to the ascii value for character '0' so that the buffer is a normal string that can be passed to printf.

18.20 ctype.h File Reference

```
#include <types.h>
```

Functions

- BOOLEAN isalpha (char c)
- BOOLEAN isupper (char c)
- BOOLEAN islower (char c)
- BOOLEAN isdigit (char c)
- BOOLEAN isspace (char c)
- char toupper (char c)
- char tolower (char c)

18.20.1 Detailed Description

Character type functions.

18.20.2 Function Documentation

18.20.2.1 isalpha()

```
BOOLEAN isalpha (
    char c )
```

Returns TRUE if the character **c** is a letter (a-z, A-Z), otherwise FALSE

Parameters

c	Character to test

18.20.2.2 isupper()

```
BOOLEAN isupper (
```

```
char c )
```

Returns TRUE if the character **c** is an uppercase letter (A-Z), otherwise FALSE

Parameters

c	Character to test

18.20.2.3 islower() `BOOLEAN islower (`

```
char c )
```

Returns TRUE if the character **c** is a lowercase letter (a-z), otherwise FALSE

Parameters

c	Character to test

18.20.2.4 isdigit() `BOOLEAN isdigit (`

```
char c )
```

Returns TRUE if the character **c** is a digit (0-9), otherwise FALSE

Parameters

c	Character to test

18.20.2.5 isspace() `BOOLEAN isspace (`

```
char c )
```

Returns TRUE if the character **c** is a space (' '), tab (\t), or newline (\n) character, otherwise FALSE

Parameters

c	Character to test

18.20.2.6 toupper() `char toupper (`

```
char c )
```

Returns uppercase version of character **c** if it is a letter (a-z), otherwise it returns the input value unchanged.

Parameters

c	Character to test

18.20.2.7 tolower() `char tolower (`

```
char c )
```

Returns lowercase version of character **c** if it is a letter (A-Z), otherwise it returns the input value unchanged.

Parameters

c	Character to test

18.21 gb/bgb_emu.h File Reference

Macros

- #define BGB_MESSAGE(message_text) BGB_MESSAGE1(BGB_ADD_DOLLARD(__LINE__), message_text)
- #define BGB_MESSAGE_FMT(buf, ...) sprintf(buf, __VA_ARGS__);BGB_MESSAGE2(BGB_ADD_DOLLARD(__LINE__), BGB_MAKE_LABEL(_##buf));
- #define BGB_PROFILE_BEGIN(MSG) BGB_MESSAGE(BGB_CONCAT(MSG,%ZEROCLKS%));
- #define BGB_PROFILE_END(MSG) BGB_MESSAGE(BGB_CONCAT(MSG,%-8+LASTCLKS%));
- #define BGB_TEXT(MSG) BGB_MESSAGE(BGB_STR(MSG))

Functions

- void BGB_profiler_message ()

18.21.1 Detailed Description

Debug window logging and profiling support for the BGB emulator.
Also see the `bgb_debug` example project included with gbdk.
See the BGB Manual for more information ("expressions, breakpoint conditions, and debug messages") http://bgb.bircd.org/manual.html#expressions

18.21.2 Macro Definition Documentation

18.21.2.1 BGB_MESSAGE `#define BGB_MESSAGE( message_text ) BGB_MESSAGE1(BGB_ADD_DOLLARD(__LINE__), message_text)`

Macro to display a message in the BGB emulator debug message window

Parameters

message_text	Quoted text string to display in the debug message window

The following special parameters can be used when bracketed with "%" characters.

- CPU registers: AF, BC, DE, HL, SP, PC, B, C, D, E, H, L, A, ZERO, ZF, Z, CARRY, CY, IME, ALLREGS
- Other state values: ROMBANK, XRAMBANK, SRAMBANK, WRAMBANK, VRAMBANK, TOTALCLKS, LASTCLKS, CLKS2VBLANK

Example: print a message along with the currently active ROM bank.

```
BGB_MESSAGE("Current ROM Bank is: %ROMBANK%");
```

See the BGB Manual for more information ("expressions, breakpoint conditions, and debug messages") http://bgb.bircd.org/manual.html#expressions

See also

BGB_PROFILE_BEGIN(), BGB_PROFILE_END()

18.21.2.2 BGB_MESSAGE_FMT `#define BGB_MESSAGE_FMT( buf, ... ) sprintf(buf, __VA_ARGS__);BGB_MESSAGE2(BGB_ADD_DOLLARD(__LINE__), BGB_MAKE_LABEL(_##buf));`

Macro to display a sprintf formatted message in the BGB emulator debug message window

Parameters

buf	Pointer to a globally defined char buffer
...	VA Args list of sprintf parameters

To avoid buffer overflows **buf** must be large enough to store the entire printed message.
Example:
```
char mybuf[100]; // should be globally defined
BGB_MESSAGE_FMT(mybuf, "An integer:%d, a string: %s", 12345, "hello bgb")
```

See also

BGB_MESSAGE()

18.21.2.3 BGB_PROFILE_BEGIN `#define BGB_PROFILE_BEGIN( MSG ) BGB_MESSAGE(BGB_CONCAT(MSG,%ZEROCLKS%));`

Macro to **Start** a profiling block for the BGB emulator

Parameters

MSG	Quoted text string to display in the debug message window

To complete the profiling block and print the result call BGB_PROFILE_END.

See also

BGB_PROFILE_END(), BGB_MESSAGE()

18.21.2.4 BGB_PROFILE_END `#define BGB_PROFILE_END( MSG ) BGB_MESSAGE(BGB_CONCAT(MSG,%-8+LASTCLKS%));`

Macro to **End** a profiling block and print the results in the BGB emulator debug message window

Parameters

MSG	Quoted text string to display in the debug message window along with the result

This should only be called after a previous call to BGB_PROFILE_BEGIN()
The results are in BGB clock units, which are "1 nop in [CGB] doublespeed mode".
So when running in Normal Speed mode (i.e. non-CGB doublespeed) the printed result should be **divided by 2** to get the actual ellapsed cycle count.
If running in CB Double Speed mode use the below call instead, it correctly compensates for the speed difference. In this scenario, the result does **not need to be divided by 2** to get the ellapsed cycle count.
```
BGB_MESSAGE("NOP TIME: %-4+LASTCLKS%");
```

See also

BGB_PROFILE_BEGIN(), BGB_MESSAGE()

18.21.2.5 BGB_TEXT `#define BGB_TEXT( MSG ) BGB_MESSAGE(BGB_STR(MSG))`

18.21.3 Function Documentation

18.21.3.1 BGB_profiler_message() `void BGB_profiler_message ( )`

Display preset debug information in the BGB debug messages window.
This function is equivalent to:
```
BGB_MESSAGE("PROFILE,%(SP+$0)%,%(SP+$1)%,%A%,%TOTALCLKS%,%ROMBANK%,%WRAMBANK%");
```

18.22 gb/cgb.h File Reference

```
#include <types.h>
```

Macros

- #define RGB(r, g, b) ((((UINT16)(b) & 0x1f) << 10) | (((UINT16)(g) & 0x1f) << 5) | (((UINT16)(r) & 0x1f) << 0))
- #define RGB_RED RGB(31, 0, 0)
- #define RGB_DARKRED RGB(15, 0, 0)
- #define RGB_GREEN RGB(0, 31, 0)
- #define RGB_DARKGREEN RGB(0, 15, 0)
- #define RGB_BLUE RGB(0, 0, 31)
- #define RGB_DARKBLUE RGB(0, 0, 15)
- #define RGB_YELLOW RGB(31, 31, 0)
- #define RGB_DARKYELLOW RGB(21, 21, 0)
- #define RGB_CYAN RGB(0, 31, 31)
- #define RGB_AQUA RGB(28, 5, 22)
- #define RGB_PINK RGB(11, 0, 31)
- #define RGB_PURPLE RGB(21, 0, 21)
- #define RGB_BLACK RGB(0, 0, 0)
- #define RGB_DARKGRAY RGB(10, 10, 10)
- #define RGB_LIGHTGRAY RGB(21, 21, 21)
- #define RGB_WHITE RGB(31, 31, 31)
- #define RGB_LIGHTFLESH RGB(30, 20, 15)
- #define RGB_BROWN RGB(10, 10, 0)
- #define RGB_ORANGE RGB(30, 20, 0)
- #define RGB_TEAL RGB(15, 15, 0)

Functions

- void set_bkg_palette (UINT8 first_palette, UINT8 nb_palettes, UINT16 *rgb_data) NONBANKED
- void set_sprite_palette (UINT8 first_palette, UINT8 nb_palettes, UINT16 *rgb_data) NONBANKED
- void set_bkg_palette_entry (UINT8 palette, UINT8 entry, UINT16 rgb_data)
- void set_sprite_palette_entry (UINT8 palette, UINT8 entry, UINT16 rgb_data)
- void cpu_slow (void)
- void cpu_fast (void)
- void cgb_compatibility (void)

18.22.1 Detailed Description

Support for the Color GameBoy (CGB).
Enabling CGB features
To unlock and use CGB features and registers you need to change byte 0143h in the cartridge header. Otherwise, the CGB will operate in monochrome "Non CGB" compatibility mode.

- Use a value of **80h** for games that support CGB and monochrome gameboys
  (with Lcc: **-Wm-yc**, or makebin directly: **-yc**)
- Use a value of **C0h** for CGB only games.
  (with Lcc: **-Wm-yC**, or makebin directly: **-yC**)

See the Pan Docs for more information CGB features.

18.22.2 Macro Definition Documentation

18.22.2.1 RGB `#define RGB(`

```
r,
g,
b ) ((((UINT16)(b) & 0x1f) << 10) | (((UINT16)(g) & 0x1f) << 5) | (((UINT16)(r)
& 0x1f) << 0))
```

Macro to create a CGB palette color entry out of the color components.

Parameters

r	Red Component, range 0 - 31 (31 brightest)
g	Green Component, range 0 - 31 (31 brightest)
b	Blue Component, range 0 - 31 (31 brightest)

The resulting format is BGR 15bpp.

See also

set_bkg_palette(), set_sprite_palette()

18.22.2.2 RGB_RED `#define RGB_RED RGB(31, 0, 0)`

Common colors based on the EGA default palette.

18.22.2.3 RGB_DARKRED `#define RGB_DARKRED RGB(15, 0, 0)`

18.22.2.4 RGB_GREEN `#define RGB_GREEN RGB( 0, 31, 0)`

18.22.2.5 RGB_DARKGREEN `#define RGB_DARKGREEN RGB( 0, 15, 0)`

18.22.2.6 RGB_BLUE `#define RGB_BLUE RGB( 0, 0, 31)`

18.22.2.7 RGB_DARKBLUE `#define RGB_DARKBLUE RGB( 0, 0, 15)`

18.22.2.8 RGB_YELLOW `#define RGB_YELLOW RGB(31, 31, 0)`

18.22.2.9 RGB_DARKYELLOW `#define RGB_DARKYELLOW RGB(21, 21, 0)`

18.22.2.10 RGB_CYAN `#define RGB_CYAN RGB( 0, 31, 31)`

18.22.2.11 RGB_AQUA `#define RGB_AQUA RGB(28, 5, 22)`

18.22.2.12 RGB_PINK `#define RGB_PINK RGB(11, 0, 31)`

18.22.2.13 RGB_PURPLE `#define RGB_PURPLE RGB(21, 0, 21)`

18.22.2.14 RGB_BLACK `#define RGB_BLACK RGB( 0, 0, 0)`

18.22.2.15 RGB_DARKGRAY `#define RGB_DARKGRAY RGB(10, 10, 10)`

18.22.2.16 RGB_LIGHTGRAY `#define RGB_LIGHTGRAY RGB(21, 21, 21)`

18.22.2.17 RGB_WHITE `#define RGB_WHITE RGB(31, 31, 31)`

18.22.2.18 RGB_LIGHTFLESH `#define RGB_LIGHTFLESH RGB(30, 20, 15)`

18.22.2.19 RGB_BROWN `#define RGB_BROWN RGB(10, 10, 0)`

18.22.2.20 RGB_ORANGE `#define RGB_ORANGE RGB(30, 20, 0)`

18.22.2.21 RGB_TEAL `#define RGB_TEAL RGB(15, 15, 0)`

18.22.3 Function Documentation

18.22.3.1 set_bkg_palette() `void set_bkg_palette (`

```
UINT8 first_palette,
UINT8 nb_palettes,
UINT16 * rgb_data )
```

Set CGB background palette(s).

Parameters

first_palette	Index of the first palette to write (0-7)
nb_palettes	Number of palettes to write (1-8, max depends on first_palette)
rgb_data	Pointer to source palette data

Writes **nb_palettes** to background palette data starting at **first_palette**, Palette data is sourced from **rgb_data**.

- Each Palette is 8 bytes in size: 4 colors x 2 bytes per palette color entry.
- Each color (4 per palette) is packed as BGR 15bpp format (1:5:5:5, MSBit [15] is unused).
- Each component (R, G, B) may have values from 0 - 31 (5 bits), 31 is brightest.

See also

RGB(), set_bkg_palette_entry()

18.22.3.2 set_sprite_palette() `void set_sprite_palette (`

```
UINT8 first_palette,
UINT8 nb_palettes,
UINT16 * rgb_data )
```

Set CGB sprite palette(s).

Parameters

first_palette	Index of the first palette to write (0-7)
nb_palettes	Number of palettes to write (1-8, max depends on first_palette)
rgb_data	Pointer to source palette data

Writes **nb_palettes** to sprite palette data starting at **first_palette**, Palette data is sourced from **rgb_data**.

- Each Palette is 8 bytes in size: 4 colors x 2 bytes per palette color entry.
- Each color (4 per palette) is packed as BGR 15bpp format (1:5:5:5, MSBit [15] is unused).
- Each component (R, G, B) may have values from 0 - 31 (5 bits), 31 is brightest.

See also

RGB(), set_sprite_palette_entry()

18.22.3.3 set_bkg_palette_entry()

```
void set_bkg_palette_entry (
    UINT8 palette,
    UINT8 entry,
    UINT16 rgb_data )
```

Sets a single color in the specified CGB background palette.

Parameters

palette	Index of the palette to modify (0-7)
entry	Index of color in palette to modify (0-3)
rgb_data	New color data in BGR 15bpp format.

See also

set_bkg_palette(), RGB()

18.22.3.4 set_sprite_palette_entry()

```
void set_sprite_palette_entry (
    UINT8 palette,
    UINT8 entry,
    UINT16 rgb_data )
```

Sets a single color in the specified CGB sprite palette.

Parameters

palette	Index of the palette to modify (0-7)
entry	Index of color in palette to modify (0-3)
rgb_data	New color data in BGR 15bpp format.

See also

set_sprite_palette(), RGB()

18.22.3.5 cpu_slow()

```
void cpu_slow (
    void  )
```

Set CPU speed to slow (Normal Speed) operation.
Interrupts are temporarily disabled and then re-enabled during this call.
In this mode the CGB operates at the same speed as the DMG/Pocket/SGB models.

- You can check to see if _cpu == CGB_TYPE before using this function.

See also

cpu_fast()

18.22.3.6 cpu_fast() `void cpu_fast ( void )`

Set CPU speed to fast (CGB Double Speed) operation.
On startup the CGB operates in Normal Speed Mode and can be switched into Double speed mode (faster processing but also higher power consumption). See the Pan Docs for more information about which hardware features operate faster and which remain at Normal Speed.

- Interrupts are temporarily disabled and then re-enabled during this call.
- You can check to see if _cpu == CGB_TYPE before using this function.

See also

cpu_slow(), _cpu

18.22.3.7 cgb_compatibility() `void cgb_compatibility ( void )`

Set defaults compatible with the normal GameBoy models.
The default/first CGB palettes for sprites and backgrounds are set to a similar default appearance as on the DMG/Pocket/SGB models. (White, Light Gray, Dark Gray, Black)

- You can check to see if _cpu == CGB_TYPE before using this function.

18.23 gb/console.h File Reference

```
#include <types.h>
```

Functions

- void gotoxy (UINT8 x, UINT8 y)
- UINT8 posx (void)
- UINT8 posy (void)
- void setchar (char c)
- void cls ()

18.23.1 Detailed Description

Console functions that work like Turbo C's.
The font is 8x8, making the screen 20x18 characters.

18.23.2 Function Documentation

18.23.2.1 gotoxy() `void gotoxy (`

```
UINT8 x,
UINT8 y )
```

Move the cursor to an absolute position at **x, y**.
x and **y** have units of tiles (8 pixels per unit)

See also

setchar()

18.23.2.2 posx() `UINT8 posx (`

```
void  )
```

Returns the current X position of the cursor.

See also

gotoxy()

18.23.2.3 posy() `UINT8 posy (`

```
void  )
```

Returns the current Y position of the cursor.

See also

gotoxy()

18.23.2.4 setchar() `void setchar (`

```
char c )
```

Writes out a single character at the current cursor position.
Does not update the cursor or interpret the character.

See also

gotoxy()

18.23.2.5 cls() `void cls ( )`

Clears the screen

18.24 gb/crash_handler.h File Reference

Functions

- void __HandleCrash ()

18.24.1 Detailed Description

When crash_handler.h is included, a crash dump screen will be displayed if the CPU executes uninitalized memory (with a value of 0xFF, the opcode for RST 38). A handler is installed for RST 38 that calls __HandleCrash().

```
#include <gb/crash_handler.h>
```

Also see the `crash` example project included with gbdk.

18.24.2 Function Documentation

18.24.2.1 __HandleCrash() `void __HandleCrash ( )`

Display the crash dump screen.
See the intro for this file for more details.

18.25 gb/drawing.h File Reference

```
#include <types.h>
```

Macros

- #define GRAPHICS_WIDTH 160
- #define GRAPHICS_HEIGHT 144
- #define SOLID 0x00 /* Overwrites the existing pixels */
- #define OR 0x01 /* Performs a logical OR */
- #define XOR 0x02 /* Performs a logical XOR */
- #define AND 0x03 /* Performs a logical AND */
- #define WHITE 0
- #define LTGREY 1
- #define DKGREY 2
- #define BLACK 3
- #define M_NOFILL 0
- #define M_FILL 1
- #define SIGNED 1
- #define UNSIGNED 0

Functions

- void gprint (char *str) NONBANKED
- void gprintln (INT16 number, INT8 radix, INT8 signed_value)
- void gprintn (INT8 number, INT8 radix, INT8 signed_value)
- INT8 gprintf (char *fmt,...) NONBANKED
- void plot (UINT8 x, UINT8 y, UINT8 colour, UINT8 mode)
- void plot_point (UINT8 x, UINT8 y)
- void switch_data (UINT8 x, UINT8 y, unsigned char *src, unsigned char *dst) NONBANKED
- void draw_image (unsigned char *data) NONBANKED
- void line (UINT8 x1, UINT8 y1, UINT8 x2, UINT8 y2)
- void box (UINT8 x1, UINT8 y1, UINT8 x2, UINT8 y2, UINT8 style)
- void circle (UINT8 x, UINT8 y, UINT8 radius, UINT8 style)
- UINT8 getpix (UINT8 x, UINT8 y)
- void wrtchr (char chr)
- void gotogxy (UINT8 x, UINT8 y)
- void color (UINT8 forecolor, UINT8 backcolor, UINT8 mode)

18.25.1 Detailed Description

All Points Addressable (APA) mode drawing library.
Drawing routines originally by Pascal Felber Legendary overhall by Jon Fuge `jonny@q-continuum.↩demon.co.uk` Commenting by Michael Hope
Note: The standard text printf() and putchar() cannot be used in APA mode - use gprintf() and wrtchr() instead.
Note: Using drawing.h will cause it's custom VBL and LCD ISRs (`drawing_vbl` and `drawing_lcd`) to be installed.

Important note for the drawing API :

```
The Game Boy graphics hardware is not well suited to frame-buffer
```

style graphics such as the kind provided in `drawing.h`. Due to that, **most drawing functions (rectangles, circles, etc) will be slow** . When possible it's much faster and more efficient to work with the tiles and tile maps that the Game Boy hardware is built around.

18.25.2 Macro Definition Documentation

18.25.2.1 GRAPHICS_WIDTH `#define GRAPHICS_WIDTH 160`

Size of the screen in pixels

18.25.2.2 GRAPHICS_HEIGHT `#define GRAPHICS_HEIGHT 144`

18.25.2.3 SOLID `#define SOLID 0x00 /* Overwrites the existing pixels */`

Possible drawing modes

18.25.2.4 OR `#define OR 0x01 /* Performs a logical OR */`

18.25.2.5 XOR `#define XOR 0x02 /* Performs a logical XOR */`

18.25.2.6 AND `#define AND 0x03 /* Performs a logical AND */`

18.25.2.7 WHITE `#define WHITE 0`

Possible drawing colours

18.25.2.8 LTGREY `#define LTGREY 1`

18.25.2.9 DKGREY `#define DKGREY 2`

18.25.2.10 BLACK `#define BLACK 3`

18.25.2.11 M_NOFILL `#define M_NOFILL 0`

Possible fill styles for box() and circle()

18.25.2.12 M_FILL `#define M_FILL 1`

18.25.2.13 SIGNED `#define SIGNED 1`

Possible values for signed_value in gprintln() and gprintn()

18.25.2.14 UNSIGNED `#define UNSIGNED 0`

18.25.3 Function Documentation

18.25.3.1 gprint() `void gprint ( char * str )`

Print the string 'str' with no interpretation

See also

gotogxy()

18.25.3.2 gprintln()

```
void gprintln (
    INT16 number,
    INT8 radix,
    INT8 signed_value )
```

Print 16 bit **number** in **radix** (base) in the default font at the current text position.

Parameters

number	number to print
radix	radix (base) to print with
signed_value	should be set to SIGNED or UNSIGNED depending on whether the number is signed or not

The current position is advanced by the numer of characters printed.

See also

gotogxy()

18.25.3.3 gprintn()

```
void gprintn (
    INT8 number,
    INT8 radix,
    INT8 signed_value )
```

Print 8 bit **number** in **radix** (base) in the default font at the current text position.

See also

gprintln(), gotogxy()

18.25.3.4 gprintf()

```
INT8 gprintf (
    char * fmt,
     ...  )
```

Print the string and arguments given by **fmt** with arguments __...__

Parameters

fmt	The format string as per printf
...	params

Currently supported:

- %c (character)
- %u (int)
- %d (INT8)
- %o (INT8 as octal)
- %x (INT8 as hex)
- %s (string)

Returns

Returns the number of items printed, or -1 if there was an error.

See also

gotogxy()

18.25.3.5 plot()

```
void plot (
    UINT8 x,
    UINT8 y,
    UINT8 colour,
    UINT8 mode )
```

Old style plot - try plot_point()

18.25.3.6 plot_point()

```
void plot_point (
    UINT8 x,
    UINT8 y )
```

Plot a point in the current drawing mode and colour at **x,y**

18.25.3.7 switch_data()

```
void switch_data (
    UINT8 x,
    UINT8 y,
    unsigned char * src,
    unsigned char * dst )
```

Exchanges the tile on screen at x,y with the tile pointed by src, original tile is saved in dst. Both src and dst may be NULL - saving or copying to screen is not performed in this case.

18.25.3.8 draw_image()

```
void draw_image (
    unsigned char * data )
```

Draw a full screen image at **data**

18.25.3.9 line()

```
void line (
    UINT8 x1,
    UINT8 y1,
    UINT8 x2,
    UINT8 y2 )
```

Draw a line in the current drawing mode and colour from **x1,y1** to **x2,y2**

18.25.3.10 box()

```
void box (
    UINT8 x1,
    UINT8 y1,
    UINT8 x2,
    UINT8 y2,
    UINT8 style )
```

Draw a box (rectangle) with corners **x1,y1** and **x2,y2** using fill mode **style** (one of NOFILL or FILL)

18.25.3.11 circle()

```
void circle (
    UINT8 x,
    UINT8 y,
    UINT8 radius,
    UINT8 style )
```

Draw a circle with centre at **x,y** and **radius** using fill mode **style** (one of NOFILL or FILL)

18.25.3.12 getpix()

```
UINT8 getpix (
    UINT8 x,
    UINT8 y )
```

Returns the current colour of the pixel at **x,y**

18.25.3.13 wrtchr() `void wrtchr (`

```
char chr )
```

Prints the character **chr** in the default font at the current text position.
The current position is advanced by 1 after the character is printed.

See also

gotogxy()

18.25.3.14 gotogxy() `void gotogxy (`

```
UINT8 x,
UINT8 y )
```

Sets the current text position to **x,y**.
Note: **x** and **y** have units of tiles (8 pixels per unit)

See also

wrtchr()

18.25.3.15 color() `void color (`

```
UINT8 forecolor,
UINT8 backcolor,
UINT8 mode )
```

Set the current **foreground** colour (for pixels), **background** colour, and draw **mode**

18.26 gb/far_ptr.h File Reference

Data Structures

- union __far_ptr

Macros

- #define TO_FAR_PTR(ofs, seg) (((FAR_PTR)seg << 16) | (FAR_PTR)ofs)
- #define FAR_SEG(ptr) (((union __far_ptr *)&ptr)->segofs.seg)
- #define FAR_OFS(ptr) (((union __far_ptr *)&ptr)->segofs.ofs)
- #define FAR_FUNC(ptr, typ) ((typ)(((union __far_ptr *)&ptr)->segfn.fn))
- #define FAR_CALL(ptr, typ, ...) (__call_banked_ptr=ptr,((typ)(&__call__banked))(__VA_ARGS__))

Typedefs

- typedef unsigned long FAR_PTR

Functions

- void __call__banked ()
- long to_far_ptr (void *ofs, int seg)

Variables

- volatile FAR_PTR __call_banked_ptr
- volatile void * __call_banked_addr
- volatile unsigned char __call_banked_bank

18.26.1 Detailed Description

Far pointers include a segment (bank) selector so they are able to point to addresses (functions or data) outside of the current bank (unlike normal pointers which are not bank-aware).
See the `banks_farptr` example project included with gbdk.

Todo Add link to a discussion about banking (such as, how to assign code and variables to banks)

18.26.2 Macro Definition Documentation

18.26.2.1 TO_FAR_PTR

```
#define TO_FAR_PTR(
        ofs,
        seg ) (((FAR_PTR)seg << 16) | (FAR_PTR)ofs)
```

Macro to obtain a far pointer at compile-time

Parameters

ofs	Memory address within the given Segment (Bank)
seg	Segment (Bank) number

Returns

A far pointer (type FAR_PTR)

18.26.2.2 FAR_SEG

```
#define FAR_SEG(
        ptr ) (((union __far_ptr *)&ptr)->segofs.seg)
```

Macro to get the Segment (Bank) number of a far pointer

Parameters

ptr	A far pointer (type FAR_PTR)

Returns

Segment (Bank) of the far pointer (type unsigned int)

18.26.2.3 FAR_OFS

```
#define FAR_OFS(
        ptr ) (((union __far_ptr *)&ptr)->segofs.ofs)
```

Macro to get the Offset (address) of a far pointer

Parameters

ptr	A far pointer (type FAR_PTR)

Returns

Offset (address) of the far pointer (type void *)

18.26.2.4 FAR_FUNC

```
#define FAR_FUNC(
        ptr,
```

```
typ ) ((typ)(((union __far_ptr *)&ptr)->segfn.fn))
```

18.26.2.5 FAR_CALL

```
#define FAR_CALL(
    ptr,
    typ,
    ... ) (__call_banked_ptr=ptr,((typ)(&__call__banked))(__VA_ARGS__))
```

Macro to call a function at far pointer **ptr** of type **typ**

Parameters

ptr	Far pointer of a function to call (type FAR_PTR)
typ	Type to cast the function far pointer to.
...	VA Args list of parameters for the function

type should match the definition of the function being called. For example:

```
// A function in bank 2
#pragma bank 2
int some_function(int param1, int param2) __banked {  return 1; };
...
// Code elsewhere, such as unbanked main()
// This type declaration should match the above function
typedef int (*some_function_t)(int, int) __banked;
// Using FAR_CALL() with the above as *ptr*, *typ*, and two parameters.
result = FAR_CALL(some_function, some_function_t, 100, 50);
```

Returns

Value returned by the function (if present)

18.26.3 Typedef Documentation

18.26.3.1 FAR_PTR

```
typedef unsigned long FAR_PTR
```

Type for storing a FAR_PTR

18.26.4 Function Documentation

18.26.4.1 __call__banked()

```
void __call__banked ( )
```

18.26.4.2 to_far_ptr()

```
long to_far_ptr (
    void * ofs,
    int seg )
```

Obtain a far pointer at runtime

Parameters

ofs	Memory address within the given Segment (Bank)
seg	Segment (Bank) number

Returns

A far pointer (type FAR_PTR)

18.26.5 Variable Documentation

18.26.5.1 __call_banked_ptr `volatile FAR_PTR __call_banked_ptr`

18.26.5.2 __call_banked_addr `volatile void* __call_banked_addr`

18.26.5.3 __call_banked_bank `volatile unsigned char __call_banked_bank`

18.27 gb/font.h File Reference

`#include <gb/gb.h>`

Data Structures

- struct sfont_handle

Macros

- #define FONT_256ENCODING 0
- #define FONT_128ENCODING 1
- #define FONT_NOENCODING 2
- #define FONT_COMPRESSED 4

Typedefs

- typedef UINT16 font_t
- typedef struct sfont_handle mfont_handle
- typedef struct sfont_handle * pmfont_handle

Functions

- void font_init (void) NONBANKED
- font_t font_load (void *font) NONBANKED
- font_t font_set (font_t font_handle) NONBANKED

Variables

- UINT8 font_spect [ ]
- UINT8 font_italic [ ]
- UINT8 font_ibm [ ]
- UINT8 font_min [ ]
- UINT8 font_ibm_fixed [ ]

18.27.1 Detailed Description

Multiple font support for the GameBoy Michael Hope, 1999 michaelh@earthling.net

18.27.2 Macro Definition Documentation

18.27.2.1 FONT_256ENCODING `#define FONT_256ENCODING 0`

Various flags in the font header.

18.27.2.2 FONT_128ENCODING `#define FONT_128ENCODING 1`

18.27.2.3 FONT_NOENCODING `#define FONT_NOENCODING 2`

18.27.2.4 FONT_COMPRESSED `#define FONT_COMPRESSED 4`

18.27.3 Typedef Documentation

18.27.3.1 font_t `typedef UINT16 font_t`

font_t is a handle to a font loaded by font_load(). It can be used with font_set()

18.27.3.2 mfont_handle `typedef struct sfont_handle mfont_handle`

Internal representation of a font. What a font_t really is

18.27.3.3 pmfont_handle `typedef struct sfont_handle* pmfont_handle`

18.27.4 Function Documentation

18.27.4.1 font_init() `void font_init ( void )`

Initializes the font system. Should be called before other font functions.

18.27.4.2 font_load() `font_t font_load ( void * font )`

Load a font and set it as the current font.

Parameters

font	Pointer to a font to load (usually a gbdk font)

Returns

Handle to the loaded font, which can be used with font_set()

See also

font_init(), font_set(), List of gbdk fonts

18.27.4.3 font_set() `font_t font_set ( font_t font_handle )`

Set the current font.

Parameters

font_handle	handle of a font returned by font_load()

Returns

The previously used font handle.

See also

font_init(), font_load()

18.28 gb/gb.h File Reference

```
#include <types.h>
#include <gb/hardware.h>
#include <gb/sgb.h>
#include <gb/cgb.h>
```

Data Structures

- struct joypads_t
- struct OAM_item_t

Macros

- #define __GBDK_VERSION 402
- #define J_START 0x80U
- #define J_SELECT 0x40U
- #define J_B 0x20U
- #define J_A 0x10U
- #define J_DOWN 0x08U
- #define J_UP 0x04U
- #define J_LEFT 0x02U
- #define J_RIGHT 0x01U
- #define M_DRAWING 0x01U
- #define M_TEXT_OUT 0x02U
- #define M_TEXT_INOUT 0x03U
- #define M_NO_SCROLL 0x04U
- #define M_NO_INTERP 0x08U
- #define S_PALETTE 0x10U
- #define S_FLIPX 0x20U
- #define S_FLIPY 0x40U
- #define S_PRIORITY 0x80U
- #define VBL_IFLAG 0x01U
- #define LCD_IFLAG 0x02U
- #define TIM_IFLAG 0x04U
- #define SIO_IFLAG 0x08U
- #define JOY_IFLAG 0x10U
- #define SCREENWIDTH 0xA0U
- #define SCREENHEIGHT 0x90U
- #define MINWNDPOSX 0x07U
- #define MINWNDPOSY 0x00U
- #define MAXWNDPOSX 0xA6U
- #define MAXWNDPOSY 0x8FU
- #define DMG_TYPE 0x01
- #define MGB_TYPE 0xFF
- #define CGB_TYPE 0x11
- #define IO_IDLE 0x00U
- #define IO_SENDING 0x01U
- #define IO_RECEIVING 0x02U
- #define IO_ERROR 0x04U
- #define SWITCH_ROM_MBC1(b) _current_bank = (b), *(unsigned char *)0x2000 = (b)
- #define SWITCH_RAM_MBC1(b) *(unsigned char *)0x4000 = (b)

- #define ENABLE_RAM_MBC1 *(unsigned char *)0x0000 = 0x0A
- #define DISABLE_RAM_MBC1 *(unsigned char *)0x0000 = 0x00
- #define SWITCH_16_8_MODE_MBC1 *(unsigned char *)0x6000 = 0x00
- #define SWITCH_4_32_MODE_MBC1 *(unsigned char *)0x6000 = 0x01
- #define SWITCH_ROM_MBC5(b)
- #define SWITCH_ROM_MBC5_8M(b)
- #define SWITCH_RAM_MBC5(b) *(unsigned char *)0x4000 = (b)
- #define ENABLE_RAM_MBC5 *(unsigned char *)0x0000 = 0x0A
- #define DISABLE_RAM_MBC5 *(unsigned char *)0x0000 = 0x00
- #define DISPLAY_ON LCDC_REG|=0x80U
- #define DISPLAY_OFF display_off();
- #define SHOW_BKG LCDC_REG|=0x01U
- #define HIDE_BKG LCDC_REG&=0xFEU
- #define SHOW_WIN LCDC_REG|=0x20U
- #define HIDE_WIN LCDC_REG&=0xDFU
- #define SHOW_SPRITES LCDC_REG|=0x02U
- #define HIDE_SPRITES LCDC_REG&=0xFDU
- #define SPRITES_8x16 LCDC_REG|=0x04U
- #define SPRITES_8x8 LCDC_REG&=0xFBU
- #define DISABLE_OAM_DMA _shadow_OAM_base = 0
- #define ENABLE_OAM_DMA _shadow_OAM_base = (UBYTE)((UWORD)&shadow_OAM >> 8)

Typedefs

- typedef void(* int_handler) (void) NONBANKED
- typedef struct OAM_item_t OAM_item_t

Functions

- void remove_VBL (int_handler h) NONBANKED
- void remove_LCD (int_handler h) NONBANKED
- void remove_TIM (int_handler h) NONBANKED
- void remove_SIO (int_handler h) NONBANKED
- void remove_JOY (int_handler h) NONBANKED
- void add_VBL (int_handler h) NONBANKED
- void add_LCD (int_handler h) NONBANKED
- void add_TIM (int_handler h) NONBANKED
- void add_SIO (int_handler h) NONBANKED
- void add_JOY (int_handler h) NONBANKED
- void nowait_int_handler (void) NONBANKED
- void wait_int_handler (void) NONBANKED
- void mode (UINT8 m) NONBANKED
- UINT8 get_mode (void) NONBANKED __preserves_regs(b
- void send_byte (void)
- void receive_byte (void)
- void delay (UINT16 d) NONBANKED
- UINT8 joypad (void) NONBANKED __preserves_regs(b
- UINT8 waitpad (UINT8 mask) NONBANKED __preserves_regs(b
- void waitpadup (void) NONBANKED __preserves_regs(a
- UINT8 joypad_init (UINT8 npads, joypads_t *joypads)
- void joypad_ex (joypads_t *joypads)
- void enable_interrupts (void) NONBANKED __preserves_regs(a
- void disable_interrupts (void) NONBANKED __preserves_regs(a
- void set_interrupts (UINT8 flags) NONBANKED __preserves_regs(b
- void reset (void) NONBANKED

- void wait_vbl_done (void) NONBANKED __preserves_regs(b
- void display_off (void) NONBANKED __preserves_regs(b
- void hiramcpy (UINT8 dst, const void *src, UINT8 n) NONBANKED __preserves_regs(b
- void set_vram_byte (UBYTE *addr, UINT8 v) __preserves_regs(b
- UINT8 * get_bkg_xy_addr (UINT8 x, UINT8 y) __preserves_regs(b
- void set_bkg_data (UINT8 first_tile, UINT8 nb_tiles, const unsigned char *data) NONBANKED __preserves_regs(b
- void set_bkg_1bit_data (UINT8 first_tile, UINT8 nb_tiles, const unsigned char *data, UINT8 color) NONBANKED __preserves_regs(b
- void get_bkg_data (UINT8 first_tile, UINT8 nb_tiles, unsigned char *data) NONBANKED __preserves_regs(b
- void set_bkg_tiles (UINT8 x, UINT8 y, UINT8 w, UINT8 h, const unsigned char *tiles) NONBANKED __preserves_regs(b
- void set_bkg_submap (UINT8 x, UINT8 y, UINT8 w, UINT8 h, const unsigned char *map, UINT8 map_w)
- void get_bkg_tiles (UINT8 x, UINT8 y, UINT8 w, UINT8 h, unsigned char *tiles) NONBANKED __preserves_regs(b
- UINT8 * set_bkg_tile_xy (UBYTE x, UBYTE y, UBYTE t) __preserves_regs(b
- void move_bkg (UINT8 x, UINT8 y)
- void scroll_bkg (INT8 x, INT8 y)
- UINT8 * get_win_xy_addr (UINT8 x, UINT8 y) __preserves_regs(b
- void set_win_data (UINT8 first_tile, UINT8 nb_tiles, const unsigned char *data) NONBANKED __preserves_regs(b
- void set_win_1bit_data (UINT8 first_tile, UINT8 nb_tiles, const unsigned char *data) NONBANKED __preserves_regs(b
- void get_win_data (UINT8 first_tile, UINT8 nb_tiles, unsigned char *data) NONBANKED __preserves_regs(b
- void set_win_tiles (UINT8 x, UINT8 y, UINT8 w, UINT8 h, const unsigned char *tiles) NONBANKED __preserves_regs(b
- void set_win_submap (UINT8 x, UINT8 y, UINT8 w, UINT8 h, const unsigned char *map, UINT8 map_w)
- void get_win_tiles (UINT8 x, UINT8 y, UINT8 w, UINT8 h, unsigned char *tiles) NONBANKED __preserves_regs(b
- UINT8 * set_win_tile_xy (UBYTE x, UBYTE y, UBYTE t) __preserves_regs(b
- void move_win (UINT8 x, UINT8 y)
- void scroll_win (INT8 x, INT8 y)
- void set_sprite_data (UINT8 first_tile, UINT8 nb_tiles, const unsigned char *data) NONBANKED __preserves_regs(b
- void set_sprite_1bit_data (UINT8 first_tile, UINT8 nb_tiles, const unsigned char *data) NONBANKED __preserves_regs(b
- void get_sprite_data (UINT8 first_tile, UINT8 nb_tiles, unsigned char *data) NONBANKED __preserves_regs(b
- void SET_SHADOW_OAM_ADDRESS (void *address)
- void set_sprite_tile (UINT8 nb, UINT8 tile)
- UINT8 get_sprite_tile (UINT8 nb)
- void set_sprite_prop (UINT8 nb, UINT8 prop)
- UINT8 get_sprite_prop (UINT8 nb)
- void move_sprite (UINT8 nb, UINT8 x, UINT8 y)
- void scroll_sprite (UINT8 nb, INT8 x, INT8 y)
- void hide_sprite (UINT8 nb)
- void set_data (unsigned char *vram_addr, const unsigned char *data, UINT16 len) NONBANKED __preserves_regs(b
- void get_data (unsigned char *data, unsigned char *vram_addr, UINT16 len) NONBANKED __preserves_regs(b
- void set_tiles (UINT8 x, UINT8 y, UINT8 w, UINT8 h, unsigned char *vram_addr, const unsigned char *tiles) NONBANKED __preserves_regs(b
- void get_tiles (UINT8 x, UINT8 y, UINT8 w, UINT8 h, unsigned char *tiles, unsigned char *vram_addr) NONBANKED __preserves_regs(b
- void init_win (UINT8 c) NONBANKED __preserves_regs(b

- void init_bkg (UINT8 c) NONBANKED __preserves_regs(b
- void vmemset (void *s, UINT8 c, size_t n) NONBANKED __preserves_regs(b
- void fill_bkg_rect (UINT8 x, UINT8 y, UINT8 w, UINT8 h, UINT8 tile) NONBANKED __preserves_regs(b
- void fill_win_rect (UINT8 x, UINT8 y, UINT8 w, UINT8 h, UINT8 tile) NONBANKED __preserves_regs(b

Variables

- UINT8 c
- UINT8 _cpu
- volatile UINT16 sys_time
- volatile UINT8 _io_status
- volatile UINT8 _io_in
- volatile UINT8 _io_out
- __REG _current_bank
- UINT8 h
- UINT8 l
- void b
- void d
- void e
- volatile struct OAM_item_t shadow_OAM [ ]
- __REG _shadow_OAM_base

18.28.1 Detailed Description

Gameboy specific functions.

18.28.2 Macro Definition Documentation

18.28.2.1 __GBDK_VERSION `#define __GBDK_VERSION 402`

18.28.2.2 J_START `#define J_START 0x80U`

Joypad bits. A logical OR of these is used in the wait_pad and joypad functions. For example, to see if the B button is pressed try
UINT8 keys; keys = joypad(); if (keys & J_B) { ... }

See also

joypad

18.28.2.3 J_SELECT `#define J_SELECT 0x40U`

18.28.2.4 J_B `#define J_B 0x20U`

18.28.2.5 J_A `#define J_A 0x10U`

18.28.2.6 J_DOWN `#define J_DOWN 0x08U`

18.28.2.7 J_UP `#define J_UP 0x04U`

18.28.2.8 J_LEFT `#define J_LEFT 0x02U`

18.28.2.9 J_RIGHT `#define J_RIGHT 0x01U`

18.28.2.10 M_DRAWING `#define M_DRAWING 0x01U`
Screen modes. Normally used by internal functions only.

See also

mode()

18.28.2.11 M_TEXT_OUT `#define M_TEXT_OUT 0x02U`

18.28.2.12 M_TEXT_INOUT `#define M_TEXT_INOUT 0x03U`

18.28.2.13 M_NO_SCROLL `#define M_NO_SCROLL 0x04U`
Set this in addition to the others to disable scrolling
If scrolling is disabled, the cursor returns to (0,0)

See also

mode()

18.28.2.14 M_NO_INTERP `#define M_NO_INTERP 0x08U`
Set this to disable interpretation

See also

mode()

18.28.2.15 S_PALETTE `#define S_PALETTE 0x10U`
If this is set, sprite colours come from OBJ1PAL. Else they come from OBJ0PAL

See also

set_sprite_prop().

18.28.2.16 S_FLIPX `#define S_FLIPX 0x20U`
If set the sprite will be flipped horizontally.

See also

set_sprite_prop()

18.28.2.17 S_FLIPY `#define S_FLIPY 0x40U`

If set the sprite will be flipped vertically.

See also

set_sprite_prop()

18.28.2.18 S_PRIORITY `#define S_PRIORITY 0x80U`

If this bit is clear, then the sprite will be displayed on top of the background and window.

See also

set_sprite_prop()

18.28.2.19 VBL_IFLAG `#define VBL_IFLAG 0x01U`

VBlank Interrupt occurs at the start of the vertical blank.
During this period the video ram may be freely accessed.

See also

set_interrupts(),

add_VBL

18.28.2.20 LCD_IFLAG `#define LCD_IFLAG 0x02U`

LCD Interrupt when triggered by the STAT register.

See also

set_interrupts(),

add_LCD

18.28.2.21 TIM_IFLAG `#define TIM_IFLAG 0x04U`

Timer Interrupt when the timer TIMA_REG overflows.

See also

set_interrupts(),

add_TIM

18.28.2.22 SIO_IFLAG `#define SIO_IFLAG 0x08U`

Serial Link Interrupt occurs when the serial transfer has completed.

See also

set_interrupts(),

add_SIO

18.28.2.23 JOY_IFLAG `#define JOY_IFLAG 0x10U`

Joypad Interrupt occurs on a transition of the keypad.

See also

set_interrupts(),

add_JOY

18.28.2.24 SCREENWIDTH `#define SCREENWIDTH 0xA0U`

Width of the visible screen in pixels.

18.28.2.25 SCREENHEIGHT `#define SCREENHEIGHT 0x90U`

Height of the visible screen in pixels.

18.28.2.26 MINWNDPOSX `#define MINWNDPOSX 0x07U`

The Minimum X position of the Window Layer (Left edge of screen)

See also

move_win()

18.28.2.27 MINWNDPOSY `#define MINWNDPOSY 0x00U`

The Minimum Y position of the Window Layer (Top edge of screen)

See also

move_win()

18.28.2.28 MAXWNDPOSX `#define MAXWNDPOSX 0xA6U`

The Maximum X position of the Window Layer (Right edge of screen)

See also

move_win()

18.28.2.29 MAXWNDPOSY `#define MAXWNDPOSY 0x8FU`

The Maximum Y position of the Window Layer (Bottom edge of screen)

See also

move_win()

18.28.2.30 DMG_TYPE `#define DMG_TYPE 0x01`

Hardware Model: Original GB or Super GB.

See also

_cpu

18.28.2.31 MGB_TYPE `#define MGB_TYPE 0xFF`

Hardware Model: Pocket GB or Super GB 2.

See also

_cpu

18.28.2.32 CGB_TYPE `#define CGB_TYPE 0x11`

Hardware Model: Color GB.

See also

_cpu

18.28.2.33 IO_IDLE `#define IO_IDLE 0x00U`

Serial Link IO is completed

18.28.2.34 IO_SENDING `#define IO_SENDING 0x01U`

Serial Link Sending data

18.28.2.35 IO_RECEIVING `#define IO_RECEIVING 0x02U`

Serial Link Receiving data

18.28.2.36 IO_ERROR `#define IO_ERROR 0x04U`

Serial Link Error

18.28.2.37 SWITCH_ROM_MBC1 `#define SWITCH_ROM_MBC1(`

`b ) _current_bank = (b), *(unsigned char *)0x2000 = (b)`

Makes MBC1 and other compatible MBCs switch the active ROM bank

Parameters

b	ROM bank to switch to

18.28.2.38 SWITCH_RAM_MBC1 `#define SWITCH_RAM_MBC1(`

`b ) *(unsigned char *)0x4000 = (b)`

Switches SRAM bank on MBC1 and other compaticle MBCs

Parameters

b	SRAM bank to switch to

18.28.2.39 ENABLE_RAM_MBC1 `#define ENABLE_RAM_MBC1 *(unsigned char *)0x0000 = 0x0A`

Enables SRAM on MBC1

18.28.2.40 DISABLE_RAM_MBC1 `#define DISABLE_RAM_MBC1 *(unsigned char *)0x0000 = 0x00`

Disables SRAM on MBC1

18.28.2.41 SWITCH_16_8_MODE_MBC1 `#define SWITCH_16_8_MODE_MBC1 *(unsigned char *)0x6000 = 0x00`

18.28.2.42 SWITCH_4_32_MODE_MBC1 `#define SWITCH_4_32_MODE_MBC1 *(unsigned char *)0x6000 = 0x01`

18.28.2.43 SWITCH_ROM_MBC5 `#define SWITCH_ROM_MBC5(b )`

Value:

```
_current_bank = (b), \
*(unsigned char *)0x3000 = 0, \
*(unsigned char *)0x2000 = (b)
```

Makes MBC5 switch to the active ROM bank; only 4M roms are supported,

See also

SWITCH_ROM_MBC5_8M()

Parameters

b	ROM bank to switch to

Note the order used here. Writing the other way around on a MBC1 always selects bank 1

18.28.2.44 SWITCH_ROM_MBC5_8M `#define SWITCH_ROM_MBC5_8M(b )`

Value:

```
*(unsigned char *)0x3000 = ((UINT16)(b) » 8), \
*(unsigned char *)0x2000 = (b)
```

Makes MBC5 to switch the active ROM bank; active bank number is not tracked by _current_bank if you use this macro

See also

_current_bank

Parameters

b	ROM bank to switch to

Note the order used here. Writing the other way around on a MBC1 always selects bank 1

18.28.2.45 SWITCH_RAM_MBC5 `#define SWITCH_RAM_MBC5(b ) *(unsigned char *)0x4000 = (b)`

Switches SRAM bank on MBC5

Parameters

b	SRAM bank to switch to

18.28.2.46 ENABLE_RAM_MBC5 `#define ENABLE_RAM_MBC5 *(unsigned char *)0x0000 = 0x0A`

Enables SRAM on MBC5

18.28.2.47 DISABLE_RAM_MBC5 `#define DISABLE_RAM_MBC5 *(unsigned char *)0x0000 = 0x00`

Disables SRAM on MBC5

18.28.2.48 DISPLAY_ON `#define DISPLAY_ON LCDC_REG|=0x80U`

Turns the display back on.

See also

display_off, DISPLAY_OFF

18.28.2.49 DISPLAY_OFF `#define DISPLAY_OFF display_off();`

Turns the display off immediatly.

See also

display_off, DISPLAY_ON

18.28.2.50 SHOW_BKG `#define SHOW_BKG LCDC_REG|=0x01U`

Turns on the background layer. Sets bit 0 of the LCDC register to 1.

18.28.2.51 HIDE_BKG `#define HIDE_BKG LCDC_REG&=0xFEU`

Turns off the background layer. Sets bit 0 of the LCDC register to 0.

18.28.2.52 SHOW_WIN `#define SHOW_WIN LCDC_REG|=0x20U`

Turns on the window layer Sets bit 5 of the LCDC register to 1.

18.28.2.53 HIDE_WIN `#define HIDE_WIN LCDC_REG&=0xDFU`

Turns off the window layer. Clears bit 5 of the LCDC register to 0.

18.28.2.54 SHOW_SPRITES `#define SHOW_SPRITES LCDC_REG|=0x02U`

Turns on the sprites layer. Sets bit 1 of the LCDC register to 1.

18.28.2.55 HIDE_SPRITES `#define HIDE_SPRITES LCDC_REG&=0xFDU`

Turns off the sprites layer. Clears bit 1 of the LCDC register to 0.

18.28.2.56 SPRITES_8x16 `#define SPRITES_8x16 LCDC_REG|=0x04U`

Sets sprite size to 8x16 pixels, two tiles one above the other. Sets bit 2 of the LCDC register to 1.

18.28.2.57 SPRITES_8x8 `#define SPRITES_8x8 LCDC_REG&=0xFBU`

Sets sprite size to 8x8 pixels, one tile. Clears bit 2 of the LCDC register to 0.

18.28.2.58 DISABLE_OAM_DMA `#define DISABLE_OAM_DMA _shadow_OAM_base = 0`

Disable OAM DMA copy each VBlank

18.28.2.59 ENABLE_OAM_DMA `#define ENABLE_OAM_DMA _shadow_OAM_base = (UBYTE)((UWORD)&shadow_OAM >> 8)`

Enable OAM DMA copy each VBlank and set it to transfer default shadow_OAM array

18.28.3 Typedef Documentation

18.28.3.1 int_handler `typedef void(* int_handler) (void) NONBANKED`

Interrupt handlers

18.28.3.2 OAM_item_t `typedef struct OAM_item_t OAM_item_t`

Sprite Attributes structure

Parameters

x	X Coordinate of the sprite on screen
y	Y Coordinate of the sprite on screen
tile	Sprite tile number (see set_sprite_tile)
prop	OAM Property Flags (see set_sprite_prop)

18.28.4 Function Documentation

18.28.4.1 remove_VBL() `void remove_VBL (`

```
int_handler h )
```

The remove functions will remove any interrupt handler.
A handler of NULL will cause bad things to happen if the given interrupt is enabled.
Removes the VBL interrupt handler.

See also

add_VBL()

18.28.4.2 remove_LCD() `void remove_LCD (`

```
int_handler h )
```

Removes the LCD interrupt handler.

See also

add_LCD(), remove_VBL()

18.28.4.3 remove_TIM() `void remove_TIM (`

```
int_handler h )
```

Removes the TIM interrupt handler.

See also

add_TIM(), remove_VBL()

18.28.4.4 remove_SIO() `void remove_SIO (`

```
int_handler h )
```

Removes the Serial Link / SIO interrupt handler.

See also

add_SIO(),

remove_VBL()

The default SIO ISR gets installed automatically if any of the standard SIO calls are used. These calls include add_SIO(), remove_SIO(), send_byte(), receive_byte().
The default SIO ISR cannot be removed once installed. Only secondary chained SIO ISRs (added with add_SIO()) can be removed.

18.28.4.5 remove_JOY() `void remove_JOY (`

`int_handler h )`

Removes the JOY interrupt handler.

See also

add_JOY(), remove_VBL()

18.28.4.6 add_VBL() `void add_VBL (`

`int_handler h )`

Adds a V-blank interrupt handler.

Parameters

h	The handler to be called whenever a V-blank interrupt occurs.

Up to 4 handlers may be added, with the last added being called last. If the remove_VBL function is to be called, only three may be added.
Note: The default VBL is installed automatically.

18.28.4.7 add_LCD() `void add_LCD (`

`int_handler h )`

Adds a LCD interrupt handler.
Called when the LCD interrupt occurs, which is normally when LY_REG == LYC_REG.
From pan/k0Pa: There are various reasons for this interrupt to occur as described by the STAT_REG register ($↩FF41). One very popular reason is to indicate to the user when the video hardware is about to redraw a given LCD line. This can be useful for dynamically controlling the SCX_REG / SCY_REG registers ($FF43/$FF42) to perform special video effects.

See also

add_VBL

18.28.4.8 add_TIM() `void add_TIM (`

`int_handler h )`

Adds a timer interrupt handler.
From pan/k0Pa: This interrupt occurs when the TIMA_REG register ($FF05) changes from $FF to $00.

See also

add_VBL

set_interrupts() with TIM_IFLAG

18.28.4.9 add_SIO() `void add_SIO (`

`int_handler h )`

Adds a Serial Link transmit complete interrupt handler.
From pan/k0Pa: This interrupt occurs when a serial transfer has completed on the game link port.

See also

send_byte, receive_byte(), add_VBL()

set_interrupts() with SIO_IFLAG

18.28.4.10 add_JOY() `void add_JOY (`

`int_handler h )`

Adds a joypad button change interrupt handler.
From pan/k0Pa: This interrupt occurs on a transition of any of the keypad input lines from high to low. Due to the fact that keypad "bounce" is virtually always present, software should expect this interrupt to occur one or more times for every button press and one or more times for every button release.

See also

joypad()

18.28.4.11 nowait_int_handler() `void nowait_int_handler (`

`void )`

Interrupt handler chain terminator that does **not** wait for .STAT
You must add this handler last in every interrupt handler chain if you want to change the default interrupt handler behaviour that waits for LCD controller mode to become 1 or 0 before return from the interrupt.
Example:

```
__critical {
    add_SIO(nowait_int_handler); // Disable wait on VRAM state before returning from SIO interrupt
}
```

See also

wait_int_handler()

18.28.4.12 wait_int_handler() `void wait_int_handler (`

`void )`

Default Interrupt handler chain terminator that waits for

See also

STAT_REG and **only** returns at the BEGINNING of either Mode 0 or Mode 1.

Used by default at the end of interrupt chains to help prevent graphical glitches. The glitches are caused when an ISR interrupts a graphics operation in one mode but returns in a different mode for which that graphics operation is not allowed.

See also

nowait_int_handler()

18.28.4.13 mode() `void mode (`

`UINT8 m )`

Set the current screen mode - one of M_∗ modes
Normally used by internal functions only.

See also

M_DRAWING, M_TEXT_OUT, M_TEXT_INOUT, M_NO_SCROLL, M_NO_INTERP

18.28.4.14 get_mode() `UINT8 get_mode (`

`void )`

Returns the current mode

See also

M_DRAWING, M_TEXT_OUT, M_TEXT_INOUT, M_NO_SCROLL, M_NO_INTERP

18.28.4.15 send_byte() `void send_byte (`

`void )`

Serial Link: Send the byte in _io_out out through the serial port
Make sure to enable interrupts for the Serial Link before trying to transfer data.

See also

add_SIO(), remove_SIO()

set_interrupts() with SIO_IFLAG

18.28.4.16 receive_byte() `void receive_byte (`

`void )`

Serial Link: Receive a byte from the serial port into _io_in
Make sure to enable interrupts for the Serial Link before trying to transfer data.

See also

add_SIO(), remove_SIO()

set_interrupts() with SIO_IFLAG

18.28.4.17 delay() `void delay (`

`UINT16 d )`

Delays the given number of milliseconds. Uses no timers or interrupts, and can be called with interrupts disabled (why nobody knows :)

18.28.4.18 joypad() `UINT8 joypad (`

`void )`

Reads and returns the current state of the joypad. Follows Nintendo's guidelines for reading the pad. Return value is an OR of J_∗

See also

J_START, J_SELECT, J_A, J_B, J_UP, J_DOWN, J_LEFT, J_RIGHT

18.28.4.19 waitpad() `UINT8 waitpad (`

`UINT8 mask )`

Waits until at least one of the buttons given in mask are pressed.

Parameters

mask	Bitmask indicating which buttons to wait for

Normally only used for checking one key, but it will support many, even J_LEFT at the same time as J_RIGHT. :)
Note: Checks in a loop that doesn't HALT at all, so the CPU will be maxed out until this call returns.

See also

joypad

J_START, J_SELECT, J_A, J_B, J_UP, J_DOWN, J_LEFT, J_RIGHT

18.28.4.20 waitpadup() `void waitpadup (`

`void )`

Waits for the directional pad and all buttons to be released.
Note: Checks in a loop that doesn't HALT at all, so the CPU will be maxed out until this call returns.

18.28.4.21 joypad_init() `UINT8 joypad_init (`

```
UINT8 npads,
joypads_t * joypads )
```

Initializes joypads_t structure for polling multiple joypads (for the GB and ones connected via SGB)

Parameters

npads	number of joypads requested (1, 2 or 4)
joypads	pointer to joypads_t structure to be initialized

Only required for joypad_ex, not required for calls to regular joypad()

Returns

number of joypads avaliable

See also

joypad_ex(), joypads_t

18.28.4.22 joypad_ex() `void joypad_ex (`

```
joypads_t * joypads )
```

Polls all avaliable joypads (for the GB and ones connected via SGB)

Parameters

joypads	pointer to joypads_t structure to be filled with joypad statuses, must be previously initialized with joypad_init()

See also

joypad_init(), joypads_t

18.28.4.23 enable_interrupts() `void enable_interrupts (`

```
void )
```

Enables unmasked interrupts

See also

disable_interrupts, set_interrupts

18.28.4.24 disable_interrupts() `void disable_interrupts (`

```
void )
```

Disables interrupts.
This function may be called as many times as you like; however the first call to enable_interrupts will re-enable them.

See also

enable_interrupts, set_interrupts

18.28.4.25 set_interrupts()

```
void set_interrupts (
    UINT8 flags )
```

Clears any pending interrupts and sets the interrupt mask register IO to flags.

Parameters

flags	A logical OR of *_IFLAGS

See also

enable_interrupts(), disable_interrupts()

VBL_IFLAG, LCD_IFLAG, TIM_IFLAG, SIO_IFLAG, JOY_IFLAG

18.28.4.26 reset()

```
void reset (
    void  )
```

Performs a warm reset by reloading the CPU value then jumping to the start of crt0 (0x0150)

18.28.4.27 wait_vbl_done()

```
void wait_vbl_done (
    void  )
```

HALTs the CPU and waits for the vertical blank interrupt (VBL) to finish.
This is often used in main loops to idle the CPU at low power until it's time to start the next frame. It's also useful for syncing animation with the screen re-draw.
Warning: If the VBL interrupt is disabled, this function will never return. If the screen is off this function returns immediatly.

18.28.4.28 display_off()

```
void display_off (
    void  )
```

Turns the display off.
Waits until the VBL interrupt before turning the display off.

See also

DISPLAY_ON

18.28.4.29 hiramcpy()

```
void hiramcpy (
    UINT8 dst,
    const void * src,
    UINT8 n )
```

Copies data from somewhere in the lower address space to part of hi-ram.

Parameters

dst	Offset in high ram (0xFF00 and above) to copy to.
src	Area to copy from
n	Number of bytes to copy.

18.28.4.30 set_vram_byte()

```
void set_vram_byte (
    UBYTE * addr,
    UINT8 v )
```

Set byte in vram at given memory location

Parameters

addr	address to write to
v	value

18.28.4.31 get_bkg_xy_addr() `UINT8* get_bkg_xy_addr (`

```
UINT8 x,
UINT8 y )
```

Get address of X,Y tile of background map

18.28.4.32 set_bkg_data() `void set_bkg_data (`

```
UINT8 first_tile,
UINT8 nb_tiles,
const unsigned char * data )
```

Sets VRAM Tile Pattern data for the Background / Window

Parameters

first_tile	Index of the first tile to write
nb_tiles	Number of tiles to write
data	Pointer to (2 bpp) source tile data

Writes **nb_tiles** tiles to VRAM starting at **first_tile**, tile data is sourced from **data**. Each Tile is 16 bytes in size (8x8 pixels, 2 bits-per-pixel).
Note: Sprite Tiles 128-255 share the same memory region as Background Tiles 128-255.
GBC only: VBK_REG determines which bank of Background tile patterns are written to.

- VBK_REG=0 indicates the first bank
- VBK_REG=1 indicates the second

18.28.4.33 set_bkg_1bit_data() `void set_bkg_1bit_data (`

```
UINT8 first_tile,
UINT8 nb_tiles,
const unsigned char * data,
UINT8 color )
```

Sets VRAM Tile Pattern data for the Background / Window using 1bpp source data

Parameters

first_tile	Index of the first Tile to write
nb_tiles	Number of Tiles to write
data	Pointer to (1bpp) source Tile Pattern data
color	Color

Similar to set_bkg_data, except source data is 1 bit-per-pixel which gets expanded into 2 bits-per-pixel.
For a given bit that represent a pixel:

- 0 will be expanded into color 0
- 1 will be expanded into color 1, 2 or 3 depending on color argument

See also

SHOW_BKG, HIDE_BKG, set_bkg_tiles

18.28.4.34 get_bkg_data()

```
void get_bkg_data (
    UINT8 first_tile,
    UINT8 nb_tiles,
    unsigned char * data )
```

Copies from Background / Window VRAM Tile Pattern data into a buffer

Parameters

first_tile	Index of the first Tile to read from
nb_tiles	Number of Tiles to read
data	Pointer to destination buffer for Tile Pattern data

Copies **nb_tiles** tiles from VRAM starting at **first_tile**, Tile data is copied into **data**.
Each Tile is 16 bytes, so the buffer pointed to by **data** should be at least **nb_tiles** x 16 bytes in size.

See also

get_win_data

18.28.4.35 set_bkg_tiles()

```
void set_bkg_tiles (
    UINT8 x,
    UINT8 y,
    UINT8 w,
    UINT8 h,
    const unsigned char * tiles )
```

Sets a rectangular region of Background Tile Map.

Parameters

x	X Start position in Background Map tile coordinates. Range 0 - 31
y	Y Start position in Background Map tile coordinates. Range 0 - 31
w	Width of area to set in tiles. Range 1 - 32
h	Height of area to set in tiles. Range 1 - 32
tiles	Pointer to source tile map data

Entries are copied from map at **tiles** to the Background Tile Map starting at **x**, **y** writing across for **w** tiles and down for **h** tiles.
Use set_bkg_submap() instead when:

- Source map is wider than 32 tiles.
- Writing a width that does not match the source map width **and** more than one row high at a time.

One byte per source tile map entry.
Writes that exceed coordinate 31 on the x or y axis will wrap around to the Left and Top edges.
Note: Patterns 128-255 overlap with patterns 128-255 of the sprite Tile Pattern table.
GBC only: VBK_REG determines whether Tile Numbers or Tile Attributes get set.

- VBK_REG=0 Tile Numbers are written
- VBK_REG=1 Tile Attributes are written

GBC Tile Attributes are defined as:

- Bit 7 - Priority flag. When this is set, it puts the tile above the sprites with colour 0 being transparent.
  0: Below sprites
  1: Above sprites
  Note: SHOW_BKG needs to be set for these priorities to take place.
- Bit 6 - Vertical flip. Dictates which way up the tile is drawn vertically.
  0: Normal
  1: Flipped Vertically
- Bit 5 - Horizontal flip. Dictates which way up the tile is drawn horizontally.
  0: Normal
  1: Flipped Horizontally
- Bit 4 - Not used
- Bit 3 - Character Bank specification. Dictates from which bank of Background Tile Patterns the tile is taken.
  0: Bank 0
  1: Bank 1
- Bit 2 - See bit 0.
- Bit 1 - See bit 0.
- Bit 0 - Bits 0-2 indicate which of the 7 BKG colour palettes the tile is assigned.

See also

SHOW_BKG

set_bkg_data, set_bkg_submap

18.28.4.36 set_bkg_submap()

```
void set_bkg_submap (
    UINT8 x,
    UINT8 y,
    UINT8 w,
    UINT8 h,
    const unsigned char * map,
    UINT8 map_w )
```

Sets a rectangular area of the Background Tile Map using a sub-region from a source tile map. Useful for scrolling implementations of maps larger than 32 x 32 tiles.

Parameters

x	X Start position in Background Map tile coordinates. Range 0 - 31
y	Y Start position in Background Map tile coordinates. Range 0 - 31
w	Width of area to set in tiles. Range 1 - 255
h	Height of area to set in tiles. Range 1 - 255
map	Pointer to source tile map data
map_w	Width of source tile map in tiles. Range 1 - 255

Entries are copied from **map** to the Background Tile Map starting at **x**, **y** writing across for **w** tiles and down for **h** tiles, using **map_w** as the rowstride for the source tile map.
Use this instead of set_bkg_tiles when the source map is wider than 32 tiles or when writing a width that does not match the source map width.
One byte per source tile map entry.
Writes that exceed coordinate 31 on the x or y axis will wrap around to the Left and Top edges.

See set_bkg_tiles for setting CGB attribute maps with VBK_REG.

See also

SHOW_BKG

set_bkg_data, set_bkg_tiles, set_win_submap

18.28.4.37 get_bkg_tiles()

```
void get_bkg_tiles (
    UINT8 x,
    UINT8 y,
    UINT8 w,
    UINT8 h,
    unsigned char * tiles )
```

Copies a rectangular region of Background Tile Map entries into a buffer.

Parameters

x	X Start position in Background Map tile coordinates. Range 0 - 31
y	Y Start position in Background Map tile coordinates. Range 0 - 31
w	Width of area to copy in tiles. Range 0 - 31
h	Height of area to copy in tiles. Range 0 - 31
tiles	Pointer to destination buffer for Tile Map data

Entries are copied into **tiles** from the Background Tile Map starting at **x**, **y** reading across for **w** tiles and down for **h** tiles.
One byte per tile.
The buffer pointed to by **tiles** should be at least **x** x **y** bytes in size.

18.28.4.38 set_bkg_tile_xy()

```
UINT8* set_bkg_tile_xy (
    UBYTE x,
    UBYTE y,
    UBYTE t )
```

Set single tile t on background layer at x,y

Parameters

x	X-coordinate
y	Y-coordinate
t	tile index

Returns

returns the address of tile, so you may use faster set_vram_byte() later

18.28.4.39 move_bkg()

```
void move_bkg (
    UINT8 x,
    UINT8 y )  [inline]
```

Moves the Background Layer to the position specified in **x** and **y** in pixels.

Parameters

x	X axis screen coordinate for Left edge of the Background
y	Y axis screen coordinate for Top edge of the Background

0,0 is the top left corner of the GB screen. The Background Layer wraps around the screen, so when part of it goes off the screen it appears on the opposite side (factoring in the larger size of the Background Layer versus the screen size).
The background layer is always under the Window Layer.

See also

SHOW_BKG, HIDE_BKG

18.28.4.40 scroll_bkg()

```
void scroll_bkg (
    INT8 x,
    INT8 y )  [inline]
```

Moves the Background relative to it's current position.

Parameters

x	Number of pixels to move the Background on the **X axis** Range: -128 - 127
y	Number of pixels to move the Background on the **Y axis** Range: -128 - 127

See also

move_bkg

18.28.4.41 get_win_xy_addr()

```
UINT8* get_win_xy_addr (
    UINT8 x,
    UINT8 y )
```

Get address of X,Y tile of window map

18.28.4.42 set_win_data()

```
void set_win_data (
    UINT8 first_tile,
    UINT8 nb_tiles,
    const unsigned char * data )
```

Sets VRAM Tile Pattern data for the Window / Background

Parameters

first_tile	Index of the first tile to write
nb_tiles	Number of tiles to write
data	Pointer to (2 bpp) source Tile Pattern data.

This is the same as set_bkg_data, since the Window Layer and Background Layer share the same Tile pattern data.

See also

set_bkg_data

set_win_tiles

SHOW_WIN, HIDE_WIN

18.28.4.43 set_win_1bit_data()

```
void set_win_1bit_data (
    UINT8 first_tile,
```

```
UINT8 nb_tiles,
const unsigned char * data )
```

Sets VRAM Tile Pattern data for the Window / Background using 1bpp source data

Parameters

first_tile	Index of the first tile to write
nb_tiles	Number of tiles to write
data	Pointer to (1bpp) source Tile Pattern data

This is the same as set_bkg_1bit_data, since the Window Layer and Background Layer share the same Tile pattern data.

See also

set_bkg_data, set_bkg_1bit_data, set_win_data

18.28.4.44 get_win_data() `void get_win_data (`

```
UINT8 first_tile,
UINT8 nb_tiles,
unsigned char * data )
```

Copies from Window / Background VRAM Tile Pattern data into a buffer

Parameters

first_tile	Index of the first Tile to read from
nb_tiles	Number of Tiles to read
data	Pointer to destination buffer for Tile Pattern Data

This is the same as get_bkg_data, since the Window Layer and Background Layer share the same Tile pattern data.

See also

get_bkg_data

18.28.4.45 set_win_tiles() `void set_win_tiles (`

```
UINT8 x,
UINT8 y,
UINT8 w,
UINT8 h,
const unsigned char * tiles )
```

Sets a rectangular region of the Window Tile Map.

Parameters

x	X Start position in Window Map tile coordinates. Range 0 - 31
y	Y Start position in Window Map tile coordinates. Range 0 - 31
w	Width of area to set in tiles. Range 1 - 32
h	Height of area to set in tiles. Range 1 - 32
tiles	Pointer to source tile map data

Entries are copied from map at **tiles** to the Window Tile Map starting at **x**, **y** writing across for **w** tiles and down for **h** tiles.
Use set_win_submap() instead when:

- Source map is wider than 32 tiles.
- Writing a width that does not match the source map width **and** more than one row high at a time.

One byte per source tile map entry.
Writes that exceed coordinate 31 on the x or y axis will wrap around to the Left and Top edges.
Note: Patterns 128-255 overlap with patterns 128-255 of the sprite Tile Pattern table.
GBC only: VBK_REG determines whether Tile Numbers or Tile Attributes get set.

- VBK_REG=0 Tile Numbers are written
- VBK_REG=1 Tile Attributes are written

For more details about GBC Tile Attributes see set_bkg_tiles.

See also

SHOW_WIN, HIDE_WIN, set_win_submap, set_bkg_tiles, set_bkg_data

18.28.4.46 set_win_submap() `void set_win_submap (`

```
UINT8 x,
UINT8 y,
UINT8 w,
UINT8 h,
const unsigned char * map,
UINT8 map_w )
```

Sets a rectangular area of the Window Tile Map using a sub-region from a source tile map.

Parameters

x	X Start position in Window Map tile coordinates. Range 0 - 31
y	Y Start position in Wimdpw Map tile coordinates. Range 0 - 31
w	Width of area to set in tiles. Range 1 - 255
h	Height of area to set in tiles. Range 1 - 255
map	Pointer to source tile map data
map_w	Width of source tile map in tiles. Range 1 - 255

Entries are copied from **map** to the Window Tile Map starting at **x**, **y** writing across for **w** tiles and down for **h** tiles, using **map_w** as the rowstride for the source tile map.
Use this instead of set_win_tiles when the source map is wider than 32 tiles or when writing a width that does not match the source map width.
One byte per source tile map entry.
Writes that exceed coordinate 31 on the x or y axis will wrap around to the Left and Top edges.
GBC only: VBK_REG determines whether Tile Numbers or Tile Attributes get set.

- VBK_REG=0 Tile Numbers are written
- VBK_REG=1 Tile Attributes are written

See set_bkg_tiles for details about CGB attribute maps with VBK_REG.

See also

SHOW_WIN, HIDE_WIN, set_win_tiles, set_bkg_submap, set_bkg_tiles, set_bkg_data

18.28.4.47 get_win_tiles() `void get_win_tiles (`

```
UINT8 x,
```

```
UINT8 y,
UINT8 w,
UINT8 h,
unsigned char * tiles )
```

Copies a rectangular region of Window Tile Map entries into a buffer.

Parameters

x	X Start position in Window Map tile coordinates. Range 0 - 31
y	Y Start position in Window Map tile coordinates. Range 0 - 31
w	Width of area to copy in tiles. Range 0 - 31
h	Height of area to copy in tiles. Range 0 - 31
tiles	Pointer to destination buffer for Tile Map data

Entries are copied into **tiles** from the Window Tile Map starting at **x**, **y** reading across for **w** tiles and down for **h** tiles.
One byte per tile.
The buffer pointed to by **tiles** should be at least **x** x **y** bytes in size.

18.28.4.48 set_win_tile_xy() UINT8* set_win_tile_xy (

```
UBYTE x,
UBYTE y,
UBYTE t )
```

Set single tile t on window layer at x,y

Parameters

x	X-coordinate
y	Y-coordinate
t	tile index

Returns

returns the address of tile, so you may use faster set_vram_byte() later

18.28.4.49 move_win() void move_win (

```
UINT8 x,
UINT8 y )  [inline]
```

Moves the Window to the **x**, **y** position on the screen.

Parameters

x	X coordinate for Left edge of the Window (actual displayed location will be X - 7)
y	Y coordinate for Top edge of the Window

7,0 is the top left corner of the screen in Window coordinates. The Window is locked to the bottom right corner.
The Window is always over the Background layer.

See also

SHOW_WIN, HIDE_WIN

18.28.4.50 scroll_win() void scroll_win (

```
            INT8 x,
            INT8 y )  [inline]
```

Move the Window relative to its current position.

Parameters

x	Number of pixels to move the window on the **X axis** Range: -128 - 127
y	Number of pixels to move the window on the **Y axis** Range: -128 - 127

See also

move_win

18.28.4.51 set_sprite_data() `void set_sprite_data (`

```
            UINT8 first_tile,
            UINT8 nb_tiles,
            const unsigned char * data )
```

Sets VRAM Tile Pattern data for Sprites

Parameters

first_tile	Index of the first tile to write
nb_tiles	Number of tiles to write
data	Pointer to (2 bpp) source Tile Pattern data

Writes **nb_tiles** tiles to VRAM starting at **first_tile**, tile data is sourced from **data**. Each Tile is 16 bytes in size (8x8 pixels, 2 bits-per-pixel).
Note: Sprite Tiles 128-255 share the same memory region as Background Tiles 128-255.
GBC only: VBK_REG determines which bank of Background tile patterns are written to.

- VBK_REG=0 indicates the first bank
- VBK_REG=1 indicates the second

18.28.4.52 set_sprite_1bit_data() `void set_sprite_1bit_data (`

```
            UINT8 first_tile,
            UINT8 nb_tiles,
            const unsigned char * data )
```

Sets VRAM Tile Pattern data for Sprites using 1bpp source data

Parameters

first_tile	Index of the first tile to write
nb_tiles	Number of tiles to write
data	Pointer to (1bpp) source Tile Pattern data

Similar to set_sprite_data, except source data is 1 bit-per-pixel which gets expanded into 2 bits-per-pixel.
For a given bit that represent a pixel:

- 0 will be expanded into color 0
- 1 will be expanded into color 3

See also

SHOW_SPRITES, HIDE_SPRITES, set_sprite_tile

18.28.4.53 get_sprite_data()

```
void get_sprite_data (
    UINT8 first_tile,
    UINT8 nb_tiles,
    unsigned char * data )
```

Copies from Sprite VRAM Tile Pattern data into a buffer

Parameters

first_tile	Index of the first tile to read from
nb_tiles	Number of tiles to read
data	Pointer to destination buffer for Tile Pattern data

Copies **nb_tiles** tiles from VRAM starting at **first_tile**, tile data is copied into **data**.
Each Tile is 16 bytes, so the buffer pointed to by **data** should be at least **nb_tiles** x 16 bytes in size.

18.28.4.54 SET_SHADOW_OAM_ADDRESS()

```
void SET_SHADOW_OAM_ADDRESS (
    void * address )  [inline]
```

Enable OAM DMA copy each VBlank and set it to transfer any 256-byte aligned array

18.28.4.55 set_sprite_tile()

```
void set_sprite_tile (
    UINT8 nb,
    UINT8 tile )  [inline]
```

Sets sprite number **nb__in the OAM to display tile number __tile**.

Parameters

nb	Sprite number, range 0 - 39
tile	Selects a tile (0 - 255) from memory at 8000h - 8FFFh In CGB Mode this could be either in VRAM Bank 0 or 1, depending on Bit 3 of the OAM Attribute Flag (see set_sprite_prop)

In 8x16 mode:

- The sprite will also display the next tile (**tile** + 1) directly below (y + 8) the first tile.
- The lower bit of the tile number is ignored: the upper 8x8 tile is (**tile** & 0xFE), and the lower 8x8 tile is (**tile** | 0x01).
- See: SPRITES_8x16

18.28.4.56 get_sprite_tile()

```
UINT8 get_sprite_tile (
    UINT8 nb )  [inline]
```

Returns the tile number of sprite number **nb** in the OAM.

Parameters

nb	Sprite number, range 0 - 39

See also

set_sprite_tile for more details

18.28.4.57 set_sprite_prop()

```
void set_sprite_prop (
    UINT8 nb,
    UINT8 prop )  [inline]
```

Sets the OAM Property Flags of sprite number **nb** to those defined in **prop**.

Parameters

nb	Sprite number, range 0 - 39
prop	Property setting (see bitfield description)

The bits in **prop** represent:

- Bit 7 - Priority flag. When this is set the sprites appear behind the background and window layer.
  0: infront
  1: behind
- Bit 6 - Vertical flip. Dictates which way up the sprite is drawn vertically.
  0: normal
  1:upside down
- Bit 5 - Horizontal flip. Dictates which way up the sprite is drawn horizontally.
  0: normal
  1:back to front
- Bit 4 - DMG/Non-CGB Mode Only. Assigns either one of the two b/w palettes to the sprite.
  0: OBJ palette 0
  1: OBJ palette 1
- Bit 3 - GBC only. Dictates from which bank of Sprite Tile Patterns the tile is taken.
  0: Bank 0
  1: Bank 1
- Bit 2 - See bit 0.
- Bit 1 - See bit 0.
- Bit 0 - GBC only. Bits 0-2 indicate which of the 7 OBJ colour palettes the sprite is assigned.

18.28.4.58 get_sprite_prop()

```
UINT8 get_sprite_prop (
    UINT8 nb )  [inline]
```

Returns the OAM Property Flags of sprite number **nb**.

Parameters

nb	Sprite number, range 0 - 39

See also

set_sprite_prop for property bitfield settings

18.28.4.59 move_sprite()

```
void move_sprite (
    UINT8 nb,
```

```
UINT8 x,
UINT8 y )  [inline]
```

Moves sprite number **nb** to the **x**, **y** position on the screen.

Parameters

nb	Sprite number, range 0 - 39
x	X Position. Specifies the sprites horizontal position on the screen (minus 8). An offscreen value (X=0 or X>=168) hides the sprite, but the sprite still affects the priority ordering - a better way to hide a sprite is to set its Y-coordinate offscreen.
y	Y Position. Specifies the sprites vertical position on the screen (minus 16). An offscreen value (for example, Y=0 or Y>=160) hides the sprite.

Moving the sprite to 0,0 (or similar off-screen location) will hide it.

18.28.4.60 scroll_sprite() `void scroll_sprite (`

```
UINT8 nb,
INT8 x,
INT8 y )  [inline]
```

Moves sprite number **nb** relative to its current position.

Parameters

nb	Sprite number, range 0 - 39
x	Number of pixels to move the sprite on the **X axis** Range: -128 - 127
y	Number of pixels to move the sprite on the **Y axis** Range: -128 - 127

See also

move_sprite for more details about the X and Y position

18.28.4.61 hide_sprite() `void hide_sprite (`

```
UINT8 nb )  [inline]
```

Hides sprite number **nb** by moving it to zero position by Y.

Parameters

nb	Sprite number, range 0 - 39

18.28.4.62 set_data() `void set_data (`

```
unsigned char * vram_addr,
const unsigned char * data,
UINT16 len )
```

Copies Tile Pattern data to an address in VRAM

Parameters

vram_addr	Pointer to destination VRAM Address
data	Pointer to source buffer
len	Number of bytes to copy

Copies **len** bytes from a buffer at **data** to VRAM starting at **vram_addr**.
GBC only: VBK_REG determines which bank of Background tile patterns are written to.

- VBK_REG=0 indicates the first bank
- VBK_REG=1 indicates the second

18.28.4.63 get_data()

```
void get_data (
    unsigned char * data,
    unsigned char * vram_addr,
    UINT16 len )
```

Copies Tile Pattern data from an address in VRAM into a buffer

Parameters

vram_addr	Pointer to source VRAM Address
data	Pointer to destination buffer
len	Number of bytes to copy

Copies **len** bytes from VRAM starting at **vram_addr** into a buffer at **data**.
GBC only: VBK_REG determines which bank of Background tile patterns are written to.

- VBK_REG=0 indicates the first bank
- VBK_REG=1 indicates the second

18.28.4.64 set_tiles()

```
void set_tiles (
    UINT8 x,
    UINT8 y,
    UINT8 w,
    UINT8 h,
    unsigned char * vram_addr,
    const unsigned char * tiles )
```

Sets a rectangular region of Tile Map entries at a given VRAM Address.

Parameters

x	X Start position in Map tile coordinates. Range 0 - 31
y	Y Start position in Map tile coordinates. Range 0 - 31
w	Width of area to set in tiles. Range 1 - 32
h	Height of area to set in tiles. Range 1 - 32
vram_addr	Pointer to destination VRAM Address
tiles	Pointer to source Tile Map data

Entries are copied from **tiles** to Tile Map at address vram_addr starting at **x**, **y** writing across for **w** tiles and down for **h** tiles.
One byte per source tile map entry.
There are two 32x32 Tile Maps in VRAM at addresses 9800h-9BFFh and 9C00h-9FFFh.
GBC only: VBK_REG determines whether Tile Numbers or Tile Attributes get set.

- VBK_REG=0 Tile Numbers are written
- VBK_REG=1 Tile Attributes are written

18.28.4.65 get_tiles()

```
void get_tiles (
    UINT8 x,
    UINT8 y,
    UINT8 w,
    UINT8 h,
    unsigned char * tiles,
    unsigned char * vram_addr )
```

Copies a rectangular region of Tile Map entries from a given VRAM Address into a buffer.

Parameters

x	X Start position in Background Map tile coordinates. Range 0 - 31
y	Y Start position in Background Map tile coordinates. Range 0 - 31
w	Width of area to copy in tiles. Range 0 - 31
h	Height of area to copy in tiles. Range 0 - 31
tiles	Pointer to destination buffer for Tile Map data
vram_addr	Pointer to source VRAM Address

Entries are copied into **tiles** from the Background Tile Map starting at **x**, **y** reading across for **w** tiles and down for **h** tiles.
One byte per tile.
There are two 32x32 Tile Maps in VRAM at addresses 9800h - 9BFFh and 9C00h - 9FFFh.
The buffer pointed to by **tiles** should be at least **x** x **y** bytes in size.

18.28.4.66 init_win()

```
void init_win (
    UINT8 c )
```

Initializes the entire Window Tile Map with Tile Number **c**

Parameters

c	Tile number to fill with

Note: This function avoids writes during modes 2 & 3

18.28.4.67 init_bkg()

```
void init_bkg (
    UINT8 c )
```

Initializes the entire Background Tile Map with Tile Number **c**

Parameters

c	Tile number to fill with

Note: This function avoids writes during modes 2 & 3

18.28.4.68 vmemset()

```
void vmemset (
    void * s,
    UINT8 c,
    size_t n )
```

Fills the VRAM memory region **s** of size **n** with Tile Number **c**

Parameters

s	Start address in VRAM
c	Tile number to fill with
n	Size of memory region (in bytes) to fill

Note: This function avoids writes during modes 2 & 3

18.28.4.69 fill_bkg_rect() `void fill_bkg_rect (`

```
UINT8 x,
UINT8 y,
UINT8 w,
UINT8 h,
UINT8 tile )
```

Fills a rectangular region of Tile Map entries for the Background layer with tile.

Parameters

x	X Start position in Background Map tile coordinates. Range 0 - 31
y	Y Start position in Background Map tile coordinates. Range 0 - 31
w	Width of area to set in tiles. Range 0 - 31
h	Height of area to set in tiles. Range 0 - 31
tile	Fill value

18.28.4.70 fill_win_rect() `void fill_win_rect (`

```
UINT8 x,
UINT8 y,
UINT8 w,
UINT8 h,
UINT8 tile )
```

Fills a rectangular region of Tile Map entries for the Window layer with tile.

Parameters

x	X Start position in Window Map tile coordinates. Range 0 - 31
y	Y Start position in Window Map tile coordinates. Range 0 - 31
w	Width of area to set in tiles. Range 0 - 31
h	Height of area to set in tiles. Range 0 - 31
tile	Fill value

18.28.5 Variable Documentation

18.28.5.1 c `int c`

18.28.5.2 _cpu `UINT8 _cpu`

GB CPU type

See also

DMG_TYPE, MGB_TYPE, CGB_TYPE, cpu_fast(), cpu_slow()

18.28.5.3 sys_time `volatile UINT16 sys_time`

Global Time Counter in VBL periods (60Hz)
Increments once per Frame
Will wrap around every ~18 minutes (unsigned 16 bits = 65535 / 60 / 60 = 18.2)

18.28.5.4 _io_status volatile UINT8 _io_status

Serial Link: Current IO Status. An OR of IO_*

18.28.5.5 _io_in volatile UINT8 _io_in

Serial Link: Byte just read after calling receive_byte()

18.28.5.6 _io_out volatile UINT8 _io_out

Serial Link: Write byte to send here before calling send_byte()

18.28.5.7 _current_bank __REG _current_bank

Tracks current active ROM bank

See also

SWITCH_ROM_MBC1(), SWITCH_ROM_MBC5() This variable is updated automatically when you call SWITCH_ROM_MBC1 or SWITCH_ROM_MBC5, or call a BANKED function.

18.28.5.8 h void h

18.28.5.9 l void l

18.28.5.10 b void b

18.28.5.11 d void d

18.28.5.12 e void e

18.28.5.13 shadow_OAM volatile struct OAM_item_t shadow_OAM[]

Shadow OAM array in WRAM, that is DMA-transferred into the real OAM each VBlank

18.28.5.14 _shadow_OAM_base __REG _shadow_OAM_base

MSB of shadow_OAM address is used by OAM DMA copying routine

18.29 gb/gbdecompress.h File Reference

Functions

- void gb_decompress (const unsigned char *sour, unsigned char *dest) __preserves_regs(b
- void gb_decompress_bkg_data (UINT8 first_tile, const unsigned char *sour) __preserves_regs(b
- void gb_decompress_win_data (UINT8 first_tile, const unsigned char *sour) __preserves_regs(b
- void gb_decompress_sprite_data (UINT8 first_tile, const unsigned char *sour) __preserves_regs(b

Variables

- void c

18.29.1 Function Documentation

18.29.1.1 gb_decompress() `void gb_decompress (`

```
    const unsigned char * sour,
    unsigned char * dest )
```

gb-decompress data from sour into dest

Parameters

sour	Pointer to source gb-compressed data
dest	Pointer to destination buffer/address

See also

gb_decompress_bkg_data, gb_decompress_win_data, gb_decompress_sprite_data

18.29.1.2 gb_decompress_bkg_data() `void gb_decompress_bkg_data (`

```
    UINT8 first_tile,
    const unsigned char * sour )
```

gb-decompress background tiles into VRAM

Parameters

first_tile	Index of the first tile to write
sour	Pointer to (gb-compressed 2 bpp) source Tile Pattern data.

Note: This function avoids writes during modes 2 & 3

See also

gb_decompress_bkg, gb_decompress_win_data, gb_decompress_sprite_data

18.29.1.3 gb_decompress_win_data() `void gb_decompress_win_data (`

```
    UINT8 first_tile,
    const unsigned char * sour )
```

gb-decompress window tiles into VRAM

Parameters

first_tile	Index of the first tile to write
sour	Pointer to (gb-compressed 2 bpp) source Tile Pattern data.

This is the same as gb_decompress_bkg_data, since the Window Layer and Background Layer share the same Tile pattern data.
Note: This function avoids writes during modes 2 & 3

See also

gb_decompress, gb_decompress_bkg_data, gb_decompress_sprite_data

18.29.1.4 gb_decompress_sprite_data() `void gb_decompress_sprite_data (`

```
    UINT8 first_tile,
    const unsigned char * sour )
```

gb-decompress sprite tiles into VRAM

Parameters

first_tile	Index of the first tile to write
sour	Pointer to source compressed data

Note: This function avoids writes during modes 2 & 3

See also

gb_decompress, gb_decompress_bkg_data, gb_decompress_win_data

18.29.2 Variable Documentation

18.29.2.1 c `void c`

18.30 gb/hardware.h File Reference

```
#include <types.h>
```

Macros

- #define __REG extern volatile __sfr

Variables

- __REG P1_REG
- __REG SB_REG
- __REG SC_REG
- __REG DIV_REG
- __REG TIMA_REG
- __REG TMA_REG
- __REG TAC_REG
- __REG IF_REG
- __REG NR10_REG
- __REG NR11_REG
- __REG NR12_REG
- __REG NR13_REG
- __REG NR14_REG
- __REG NR21_REG
- __REG NR22_REG
- __REG NR23_REG
- __REG NR24_REG
- __REG NR30_REG
- __REG NR31_REG
- __REG NR32_REG
- __REG NR33_REG
- __REG NR34_REG
- __REG NR41_REG
- __REG NR42_REG
- __REG NR43_REG
- __REG NR44_REG
- __REG NR50_REG
- __REG NR51_REG
- __REG NR52_REG

- __REG LCDC_REG
- __REG STAT_REG
- __REG SCY_REG
- __REG SCX_REG
- __REG LY_REG
- __REG LYC_REG
- __REG DMA_REG
- __REG BGP_REG
- __REG OBP0_REG
- __REG OBP1_REG
- __REG WY_REG
- __REG WX_REG
- __REG KEY1_REG
- __REG VBK_REG
- __REG HDMA1_REG
- __REG HDMA2_REG
- __REG HDMA3_REG
- __REG HDMA4_REG
- __REG HDMA5_REG
- __REG RP_REG
- __REG BCPS_REG
- __REG BCPD_REG
- __REG OCPS_REG
- __REG OCPD_REG
- __REG SVBK_REG
- __REG IE_REG

18.30.1 Detailed Description

Defines that let the GB's hardware registers be accessed from C.
See the Pandocs for more details on each register.

18.30.2 Macro Definition Documentation

18.30.2.1 __REG `#define __REG extern volatile __sfr`

18.30.3 Variable Documentation

18.30.3.1 P1_REG `__REG P1_REG`
Joystick: 1.1.P15.P14.P13.P12.P11.P10

18.30.3.2 SB_REG `__REG SB_REG`
Serial IO data buffer

18.30.3.3 SC_REG `__REG SC_REG`
Serial IO control register

18.30.3.4 DIV_REG `__REG DIV_REG`
Divider register

18.30.3.5 TIMA_REG `__REG TIMA_REG`
Timer counter

18.30.3.6 TMA_REG __REG TMA_REG
Timer modulo

18.30.3.7 TAC_REG __REG TAC_REG
Timer control

18.30.3.8 IF_REG __REG IF_REG
Interrupt flags: 0.0.0.JOY.SIO.TIM.LCD.VBL

18.30.3.9 NR10_REG __REG NR10_REG
Sound Channel 1 Sweep

18.30.3.10 NR11_REG __REG NR11_REG
Sound Channel 1 Sound length/Wave pattern duty

18.30.3.11 NR12_REG __REG NR12_REG
Sound Channel 1 Volume Envelope

18.30.3.12 NR13_REG __REG NR13_REG
Sound Channel 1 Frequency Low

18.30.3.13 NR14_REG __REG NR14_REG
Sound Channel 1 Frequency High

18.30.3.14 NR21_REG __REG NR21_REG
Sound Channel 2 Tone

18.30.3.15 NR22_REG __REG NR22_REG
Sound Channel 2 Volume Envelope

18.30.3.16 NR23_REG __REG NR23_REG
Sound Channel 2 Frequency data Low

18.30.3.17 NR24_REG __REG NR24_REG
Sound Channel 2 Frequency data High

18.30.3.18 NR30_REG __REG NR30_REG
Sound Channel 3 Sound on/off

18.30.3.19 NR31_REG __REG NR31_REG
Sound Channel 3 Sound Length

18.30.3.20 NR32_REG __REG NR32_REG
Sound Channel 3 Select output level

18.30.3.21 NR33_REG __REG NR33_REG
Sound Channel 3 Frequency data Low

18.30.3.22 NR34_REG __REG NR34_REG
Sound Channel 3 Frequency data High

18.30.3.23 NR41_REG __REG NR41_REG
Sound Channel 4 Sound Length

18.30.3.24 NR42_REG __REG NR42_REG
Sound Channel 4 Volume Envelope

18.30.3.25 NR43_REG __REG NR43_REG
Sound Channel 4 Polynomial Counter

18.30.3.26 NR44_REG __REG NR44_REG
Sound Channel 4 Counter / Consecutive and Inital

18.30.3.27 NR50_REG __REG NR50_REG
Sound Channel control / ON-OFF / Volume

18.30.3.28 NR51_REG __REG NR51_REG
Sound Selection of Sound output terminal

18.30.3.29 NR52_REG __REG NR52_REG
Sound Master on/off

18.30.3.30 LCDC_REG __REG LCDC_REG
LCD control

18.30.3.31 STAT_REG __REG STAT_REG
LCD status

18.30.3.32 SCY_REG __REG SCY_REG
Scroll Y

18.30.3.33 SCX_REG __REG SCX_REG
Scroll X

18.30.3.34 LY_REG __REG LY_REG
LCDC Y-coordinate

18.30.3.35 LYC_REG __REG LYC_REG
LY compare

18.30.3.36 DMA_REG __REG DMA_REG
DMA transfer

18.30.3.37 BGP_REG __REG BGP_REG
BG palette data

18.30.3.38 OBP0_REG __REG OBP0_REG
OBJ palette 0 data

18.30.3.39 OBP1_REG __REG OBP1_REG
OBJ palette 1 data

18.30.3.40 WY_REG __REG WY_REG
Window Y coordinate

18.30.3.41 WX_REG __REG WX_REG
Window X coordinate

18.30.3.42 KEY1_REG __REG KEY1_REG

CPU speed

18.30.3.43 VBK_REG __REG VBK_REG

VRAM bank

18.30.3.44 HDMA1_REG __REG HDMA1_REG

DMA control 1

18.30.3.45 HDMA2_REG __REG HDMA2_REG

DMA control 2

18.30.3.46 HDMA3_REG __REG HDMA3_REG

DMA control 3

18.30.3.47 HDMA4_REG __REG HDMA4_REG

DMA control 4

18.30.3.48 HDMA5_REG __REG HDMA5_REG

DMA control 5

18.30.3.49 RP_REG __REG RP_REG

IR port

18.30.3.50 BCPS_REG __REG BCPS_REG

BG color palette specification

18.30.3.51 BCPD_REG __REG BCPD_REG

BG color palette data

18.30.3.52 OCPS_REG __REG OCPS_REG

OBJ color palette specification

18.30.3.53 OCPD_REG __REG OCPD_REG

OBJ color palette data

18.30.3.54 SVBK_REG __REG SVBK_REG

WRAM bank

18.30.3.55 IE_REG __REG IE_REG

Interrupt enable

18.31 gb/malloc.h File Reference

```
#include <types.h>
```

Data Structures

- struct smalloc_hunk

Macros

- #define MALLOC_FREE 1
- #define MALLOC_USED 2
- #define MALLOC_MAGIC 123

Typedefs

- typedef struct smalloc_hunk mmalloc_hunk
- typedef struct smalloc_hunk * pmmalloc_hunk

Functions

- void malloc_gc (void) NONBANKED
- void debug (char *routine, char *msg) NONBANKED

Variables

- UBYTE malloc_heap_start
- pmmalloc_hunk malloc_first

18.31.1 Detailed Description

Header for a simple implementation of malloc().

Todo : This library may currently be broken.

18.31.2 Macro Definition Documentation

18.31.2.1 MALLOC_FREE `#define MALLOC_FREE 1`

The malloc hunk flags Note: Cound have used a negative size a'la TI

18.31.2.2 MALLOC_USED `#define MALLOC_USED 2`

18.31.2.3 MALLOC_MAGIC `#define MALLOC_MAGIC 123`

Magic number of a header. Gives us some chance of surviving if the list is corrupted

18.31.3 Typedef Documentation

18.31.3.1 mmalloc_hunk `typedef struct smalloc_hunk mmalloc_hunk`

18.31.3.2 pmmalloc_hunk `typedef struct smalloc_hunk* pmmalloc_hunk`

18.31.4 Function Documentation

18.31.4.1 malloc_gc() `void malloc_gc (`

```
void  )
```

Garbage collect (join free hunks)

18.31.4.2 debug() `void debug (`

```
char * routine,
char * msg )
```

debug message logger

18.31.5 Variable Documentation

18.31.5.1 malloc_heap_start `UBYTE malloc_heap_start`

Start of free memory, as defined by the linker

18.31.5.2 malloc_first `pmmalloc_hunk malloc_first`

First hunk

18.32 gb/metasprites.h File Reference

Data Structures

- struct metasprite_t

Macros

- #define metasprite_end -128

Typedefs

- typedef struct metasprite_t metasprite_t

Functions

- UBYTE move_metasprite (const metasprite_t *metasprite, UINT8 base_tile, UINT8 base_sprite, UINT8 x, UINT8 y)
- UBYTE move_metasprite_vflip (const metasprite_t *metasprite, UINT8 base_tile, UINT8 base_sprite, UINT8 x, UINT8 y)
- UBYTE move_metasprite_hflip (const metasprite_t *metasprite, UINT8 base_tile, UINT8 base_sprite, UINT8 x, UINT8 y)
- UBYTE move_metasprite_hvflip (const metasprite_t *metasprite, UINT8 base_tile, UINT8 base_sprite, UINT8 x, UINT8 y)
- void hide_metasprite (const metasprite_t *metasprite, UINT8 base_sprite)

Variables

- const void * __current_metasprite
- UBYTE __current_base_tile
- UBYTE __render_shadow_OAM

18.32.1 Detailed Description

Metasprite support

18.32.2 Macro Definition Documentation

18.32.2.1 metasprite_end `#define metasprite_end -128`

18.32.3 Typedef Documentation

18.32.3.1 metasprite_t `typedef struct metasprite_t metasprite_t`

metasprite item description

18.32.4 Function Documentation

18.32.4.1 move_metasprite()

```
UBYTE move_metasprite (
    const metasprite_t * metasprite,
    UINT8 base_tile,
    UINT8 base_sprite,
    UINT8 x,
    UINT8 y )  [inline]
```

Moves metasprite to the absolute position x and y, allocating hardware sprites from base_sprite using tiles from base_tile

Parameters

metasprite	metasprite description
base_tile	start tile where tiles for that metasprite begin
base_sprite	start hardware sprite
x	absolute x coordinate of the sprite
y	absolute y coordinate of the sprite

Returns

number of hardware sprites used to draw this metasprite

18.32.4.2 move_metasprite_vflip()

```
UBYTE move_metasprite_vflip (
    const metasprite_t * metasprite,
    UINT8 base_tile,
    UINT8 base_sprite,
    UINT8 x,
    UINT8 y )  [inline]
```

18.32.4.3 move_metasprite_hflip()

```
UBYTE move_metasprite_hflip (
    const metasprite_t * metasprite,
    UINT8 base_tile,
    UINT8 base_sprite,
    UINT8 x,
    UINT8 y )  [inline]
```

18.32.4.4 move_metasprite_hvflip()

```
UBYTE move_metasprite_hvflip (
    const metasprite_t * metasprite,
    UINT8 base_tile,
    UINT8 base_sprite,
    UINT8 x,
    UINT8 y )  [inline]
```

18.32.4.5 hide_metasprite()

```
void hide_metasprite (
    const metasprite_t * metasprite,
    UINT8 base_sprite )  [inline]
```

Hides metasprite from screen

Parameters

metasprite	metasprite description
base_sprite	start hardware sprite

18.32.5 Variable Documentation

18.32.5.1 __current_metasprite `const void* __current_metasprite`

18.32.5.2 __current_base_tile `UBYTE __current_base_tile`

18.32.5.3 __render_shadow_OAM `UBYTE __render_shadow_OAM`

18.33 gb/sample.h File Reference

Functions

- void play_sample (UINT8 *start, UINT16 len) NONBANKED

18.33.1 Detailed Description

Playback raw sound sample with length len from start at 8192Hz rate. len defines the length of the sample in samples/32 or bytes/16. The format of the data is unsigned 4-bit samples, 2 samples per byte, upper 4-bits played before lower 4 bits.
Adaption for GBDK by Lars Malmborg. Original code by Jeff Frohwein.

18.33.2 Function Documentation

18.33.2.1 play_sample() `void play_sample (`

```
    UINT8 * start,
    UINT16 len )
```

Play the given, appropriatly formatted sample.

18.34 gb/sgb.h File Reference

Macros

- #define SGB_PAL_01 0x00U
- #define SGB_PAL_23 0x01U
- #define SGB_PAL_03 0x02U
- #define SGB_PAL_12 0x03U
- #define SGB_ATTR_BLK 0x04U
- #define SGB_ATTR_LIN 0x05U
- #define SGB_ATTR_DIV 0x06U
- #define SGB_ATTR_CHR 0x07U
- #define SGB_SOUND 0x08U
- #define SGB_SOU_TRN 0x09U
- #define SGB_PAL_SET 0x0AU
- #define SGB_PAL_TRN 0x0BU
- #define SGB_ATRC_EN 0x0CU

- #define SGB_TEST_EN 0x0DU
- #define SGB_ICON_EN 0x0EU
- #define SGB_DATA_SND 0x0FU
- #define SGB_DATA_TRN 0x10U
- #define SGB_MLT_REQ 0x11U
- #define SGB_JUMP 0x12U
- #define SGB_CHR_TRN 0x13U
- #define SGB_PCT_TRN 0x14U
- #define SGB_ATTR_TRN 0x15U
- #define SGB_ATTR_SET 0x16U
- #define SGB_MASK_EN 0x17U
- #define SGB_OBJ_TRN 0x18U

Functions

- UINT8 sgb_check (void)
- void sgb_transfer (unsigned char *packet)
- void sgb_transfer_nowait (unsigned char *packet)

18.34.1 Detailed Description

Super Gameboy definitions.
See the example SGB project for additional details.

18.34.2 Macro Definition Documentation

18.34.2.1 SGB_PAL_01 `#define SGB_PAL_01 0x00U`

SGB Command: Set SGB Palettes 0 & 1

18.34.2.2 SGB_PAL_23 `#define SGB_PAL_23 0x01U`

SGB Command: Set SGB Palettes 2 & 3

18.34.2.3 SGB_PAL_03 `#define SGB_PAL_03 0x02U`

SGB Command: Set SGB Palettes 0 & 3

18.34.2.4 SGB_PAL_12 `#define SGB_PAL_12 0x03U`

SGB Command: Set SGB Palettes 1 & 2

18.34.2.5 SGB_ATTR_BLK `#define SGB_ATTR_BLK 0x04U`

SGB Command: Set color attributes for rectangular regions

18.34.2.6 SGB_ATTR_LIN `#define SGB_ATTR_LIN 0x05U`

SGB Command: Set color attributes for horizontal or vertical character lines

18.34.2.7 SGB_ATTR_DIV `#define SGB_ATTR_DIV 0x06U`

SGB Command: Split screen in half and assign separate color attribes to each side and the divider

18.34.2.8 SGB_ATTR_CHR `#define SGB_ATTR_CHR 0x07U`

SGB Command: Set color attributes for separate charactersSet SGB Palette 0,1 Data

18.34.2.9 SGB_SOUND `#define SGB_SOUND 0x08U`

SGB Command: Start and stop a internal sound effect, and sounds using internal tone data

18.34.2.10 SGB_SOU_TRN `#define SGB_SOU_TRN 0x09U`
SGB Command: Transfer sound code or data to the SNES APU RAM

18.34.2.11 SGB_PAL_SET `#define SGB_PAL_SET 0x0AU`
SGB Command: Apply (previously transferred) SGB system color palettes to actual SNES palettes

18.34.2.12 SGB_PAL_TRN `#define SGB_PAL_TRN 0x0BU`
SGB Command: Transfer palette data into SGB system color palettes

18.34.2.13 SGB_ATRC_EN `#define SGB_ATRC_EN 0x0CU`
SGB Command: Enable/disable Attraction mode. It is enabled by default

18.34.2.14 SGB_TEST_EN `#define SGB_TEST_EN 0x0DU`
SGB Command: Enable/disable test mode for "SGB-CPU variable clock speed function"

18.34.2.15 SGB_ICON_EN `#define SGB_ICON_EN 0x0EU`
SGB Command: Enable/disable ICON functionality

18.34.2.16 SGB_DATA_SND `#define SGB_DATA_SND 0x0FU`
SGB Command: Write one or more bytes into SNES Work RAM

18.34.2.17 SGB_DATA_TRN `#define SGB_DATA_TRN 0x10U`
SGB Command: Transfer code or data into SNES RAM

18.34.2.18 SGB_MLT_REQ `#define SGB_MLT_REQ 0x11U`
SGB Command: Request multiplayer mode (input from more than one joypad)

18.34.2.19 SGB_JUMP `#define SGB_JUMP 0x12U`
SGB Command: Set the SNES program counter and NMI (vblank interrupt) handler to specific addresses

18.34.2.20 SGB_CHR_TRN `#define SGB_CHR_TRN 0x13U`
SGB Command: Transfer tile data (characters) to SNES Tile memory

18.34.2.21 SGB_PCT_TRN `#define SGB_PCT_TRN 0x14U`
SGB Command: Transfer tile map and palette data to SNES BG Map memory

18.34.2.22 SGB_ATTR_TRN `#define SGB_ATTR_TRN 0x15U`
SGB Command: Transfer data to (color) Attribute Files (ATFs) in SNES RAM

18.34.2.23 SGB_ATTR_SET `#define SGB_ATTR_SET 0x16U`
SGB Command: Transfer attributes from (color) Attribute Files (ATF) to the Game Boy window

18.34.2.24 SGB_MASK_EN `#define SGB_MASK_EN 0x17U`
SGB Command: Modify Game Boy window mask settings

18.34.2.25 SGB_OBJ_TRN `#define SGB_OBJ_TRN 0x18U`
SGB Command: Transfer OBJ attributes to SNES OAM memory

18.34.3 Function Documentation

18.34.3.1 sgb_check() `UINT8 sgb_check (void )`
Returns a non-null value if running on Super GameBoy

18.34.3.2 sgb_transfer() `void sgb_transfer ( unsigned char * packet )`

Transfer a SGB packet

Parameters

packet	Pointer to buffer with SGB packet data.

The first byte of **packet** should be a SGB command, then up to 15 bytes of command parameter data.
See the `sgb_border` GBDK example project for a demo of how to use these the sgb functions.

See also

sgb_check()

18.34.3.3 sgb_transfer_nowait() `void sgb_transfer_nowait ( unsigned char * packet )`

Transfer a SGB packet without the 60 ms / 4 frame delay at the end (the delay time is required between consecutive SGB packets)

Parameters

packet	Pointer to buffer with SGB packet data.

See also

sgb_transfer()

18.35 gbdk-lib.h File Reference

```
#include <asm/gbz80/provides.h>
```

18.35.1 Detailed Description

Settings for the greater library system.

18.36 limits.h File Reference

Macros

- #define CHAR_BIT 8 /∗ bits in a char ∗/
- #define SCHAR_MAX 127
- #define SCHAR_MIN -128
- #define UCHAR_MAX 0xff
- #define CHAR_MAX SCHAR_MAX
- #define CHAR_MIN SCHAR_MIN
- #define INT_MIN (-32767 - 1)
- #define INT_MAX 32767
- #define SHRT_MAX INT_MAX
- #define SHRT_MIN INT_MIN
- #define UINT_MAX 0xffff
- #define UINT_MIN 0
- #define USHRT_MAX UINT_MAX
- #define USHRT_MIN UINT_MIN
- #define LONG_MIN (-2147483647L-1)

- #define LONG_MAX 2147483647L
- #define ULONG_MAX 0xffffffff
- #define ULONG_MIN 0

18.36.1 Macro Definition Documentation

18.36.1.1 CHAR_BIT `#define CHAR_BIT 8 /* bits in a char */`

18.36.1.2 SCHAR_MAX `#define SCHAR_MAX 127`

18.36.1.3 SCHAR_MIN `#define SCHAR_MIN -128`

18.36.1.4 UCHAR_MAX `#define UCHAR_MAX 0xff`

18.36.1.5 CHAR_MAX `#define CHAR_MAX SCHAR_MAX`

18.36.1.6 CHAR_MIN `#define CHAR_MIN SCHAR_MIN`

18.36.1.7 INT_MIN `#define INT_MIN (-32767 - 1)`

18.36.1.8 INT_MAX `#define INT_MAX 32767`

18.36.1.9 SHRT_MAX `#define SHRT_MAX INT_MAX`

18.36.1.10 SHRT_MIN `#define SHRT_MIN INT_MIN`

18.36.1.11 UINT_MAX `#define UINT_MAX 0xffff`

18.36.1.12 UINT_MIN `#define UINT_MIN 0`

18.36.1.13 USHRT_MAX `#define USHRT_MAX UINT_MAX`

18.36.1.14 USHRT_MIN `#define USHRT_MIN UINT_MIN`

18.36.1.15 LONG_MIN `#define LONG_MIN (-2147483647L-1)`

18.36.1.16 LONG_MAX `#define LONG_MAX 2147483647L`

18.36.1.17 ULONG_MAX `#define ULONG_MAX 0xffffffff`

18.36.1.18 ULONG_MIN `#define ULONG_MIN 0`

18.37 rand.h File Reference

`#include <types.h>`

Functions

- void initrand (UINT16 seed) NONBANKED
- INT8 rand (void)
- UINT16 randw (void)
- void initarand (UINT16 seed)
- INT8 arand (void)

18.37.1 Detailed Description

Random generator using the linear congruential method

Author

Luc Van den Borre

18.37.2 Function Documentation

18.37.2.1 initrand() `void initrand ( UINT16 seed )`

Initalise the pseudo-random number generator.

Parameters

seed	The value for initializing the random number generator.

The seed should be different each time, otherwise the same pseudo-random sequence will be generated.
The DIV Register (DIV_REG) is sometimes used as a seed, particularly if read at some variable point in time (such as when the player presses a button).
Only needs to be called once to initialize, buy may be called again to re-initialize with the same or a different seed.

See also

rand(), randw()

18.37.2.2 rand() `INT8 rand ( void )`

Returns a random byte (8 bit) value.
initrand() should be used to initialize the random number generator before using rand()

18.37.2.3 randw() `UINT16 randw ( void )`

Returns a random word (16 bit) value.
initrand() should be used to initialize the random number generator before using rand()

18.37.2.4 initarand() `void initarand ( UINT16 seed )`

Random generator using the linear lagged additive method

Parameters

seed	The value for initializing the random number generator.

Note: initarand() calls initrand() with the same seed value, and uses rand() to initialize the random generator.

See also

initrand() for suggestions about seed values, arand()

18.37.2.5 arand() `INT8 arand ( void )`

Returns a random number generated with the linear lagged additive method.
initarand() should be used to initialize the random number generator before using arand()

18.38 setjmp.h File Reference

Macros

- #define SP_SIZE 1
- #define BP_SIZE 0
- #define SPX_SIZE 0
- #define BPX_SIZE SPX_SIZE
- #define RET_SIZE 2
- #define setjmp(jump_buf) __setjmp(jump_buf)

Typedefs

- typedef unsigned char jmp_buf[RET_SIZE+SP_SIZE+BP_SIZE+SPX_SIZE+BPX_SIZE]

Functions

- int __setjmp (jmp_buf)
- _Noreturn void longjmp (jmp_buf, int)

18.38.1 Macro Definition Documentation

18.38.1.1 SP_SIZE `#define SP_SIZE 1`

18.38.1.2 BP_SIZE `#define BP_SIZE 0`

18.38.1.3 SPX_SIZE `#define SPX_SIZE 0`

18.38.1.4 BPX_SIZE `#define BPX_SIZE SPX_SIZE`

18.38.1.5 RET_SIZE `#define RET_SIZE 2`

18.38.1.6 setjmp `#define setjmp(`
jump_buf `) __setjmp(jump_buf)`

18.38.2 Typedef Documentation

18.38.2.1 jmp_buf `typedef unsigned char jmp_buf[RET_SIZE+SP_SIZE+BP_SIZE+SPX_SIZE+BPX_SIZE]`

18.38.3 Function Documentation

18.38.3.1 __setjmp() `int __setjmp (`
`jmp_buf )`

18.38.3.2 longjmp() `_Noreturn void longjmp (`
`jmp_buf ,`
`int )`

18.39 stdatomic.h File Reference

Data Structures

- struct atomic_flag

Functions

- _Bool atomic_flag_test_and_set (volatile atomic_flag ∗object)
- void atomic_flag_clear (volatile atomic_flag ∗object)

18.39.1 Function Documentation

18.39.1.1 atomic_flag_test_and_set() `_Bool atomic_flag_test_and_set (`
`volatile atomic_flag * object )`

18.39.1.2 atomic_flag_clear() `void atomic_flag_clear (`
`volatile atomic_flag * object )`

18.40 stdbool.h File Reference

Macros

- #define true ((_Bool)+1)
- #define false ((_Bool)+0)
- #define bool _Bool
- #define __bool_true_false_are_defined 1

18.40.1 Macro Definition Documentation

18.40.1.1 true `#define true ((_Bool)+1)`

18.40.1.2 false `#define false ((_Bool)+0)`

18.40.1.3 bool `#define bool _Bool`

18.40.1.4 __bool_true_false_are_defined `#define __bool_true_false_are_defined 1`

18.41 stddef.h File Reference

Macros

- #define NULL (void *)0
- #define __PTRDIFF_T_DEFINED
- #define __SIZE_T_DEFINED
- #define __WCHAR_T_DEFINED
- #define offsetof(s, m) __builtin_offsetof (s, m)

Typedefs

- typedef int ptrdiff_t
- typedef unsigned int size_t
- typedef unsigned long int wchar_t

18.41.1 Macro Definition Documentation

18.41.1.1 NULL `#define NULL (void *)0`

18.41.1.2 __PTRDIFF_T_DEFINED `#define __PTRDIFF_T_DEFINED`

18.41.1.3 __SIZE_T_DEFINED `#define __SIZE_T_DEFINED`

18.41.1.4 __WCHAR_T_DEFINED `#define __WCHAR_T_DEFINED`

18.41.1.5 offsetof

```
#define offsetof(
    s,
    m ) __builtin_offsetof (s, m)
```

18.41.2 Typedef Documentation

18.41.2.1 ptrdiff_t `typedef int ptrdiff_t`

18.41.2.2 size_t `typedef unsigned int size_t`

18.41.2.3 wchar_t `typedef unsigned long int wchar_t`

18.42 stdint.h File Reference

Macros

- #define INT8_MIN (-128)
- #define INT16_MIN (-32767-1)
- #define INT32_MIN (-2147483647L-1)
- #define INT8_MAX (127)
- #define INT16_MAX (32767)
- #define INT32_MAX (2147483647L)
- #define UINT8_MAX (255)
- #define UINT16_MAX (65535)
- #define UINT32_MAX (4294967295UL)
- #define INT_LEAST8_MIN INT8_MIN
- #define INT_LEAST16_MIN INT16_MIN
- #define INT_LEAST32_MIN INT32_MIN
- #define INT_LEAST8_MAX INT8_MAX
- #define INT_LEAST16_MAX INT16_MAX
- #define INT_LEAST32_MAX INT32_MAX
- #define UINT_LEAST8_MAX UINT8_MAX
- #define UINT_LEAST16_MAX UINT16_MAX
- #define UINT_LEAST32_MAX UINT32_MAX
- #define INT_FAST8_MIN INT8_MIN
- #define INT_FAST16_MIN INT16_MIN
- #define INT_FAST32_MIN INT32_MIN
- #define INT_FAST8_MAX INT8_MAX
- #define INT_FAST16_MAX INT16_MAX
- #define INT_FAST32_MAX INT32_MAX
- #define UINT_FAST8_MAX UINT8_MAX
- #define UINT_FAST16_MAX UINT16_MAX
- #define UINT_FAST32_MAX UINT32_MAX
- #define INTPTR_MIN (-32767-1)
- #define INTPTR_MAX (32767)
- #define UINTPTR_MAX (65535)
- #define INTMAX_MIN (-2147483647L-1)
- #define INTMAX_MAX (2147483647L)
- #define UINTMAX_MAX (4294967295UL)
- #define PTRDIFF_MIN (-32767-1)
- #define PTRDIFF_MAX (32767)
- #define SIG_ATOMIC_MIN (0)
- #define SIG_ATOMIC_MAX (255)
- #define SIZE_MAX (65535u)
- #define INT8_C(c) c
- #define INT16_C(c) c
- #define INT32_C(c) c ## L
- #define UINT8_C(c) c ## U
- #define UINT16_C(c) c ## U
- #define UINT32_C(c) c ## UL
- #define WCHAR_MIN 0
- #define WCHAR_MAX 0xffffffff
- #define WINT_MIN 0
- #define WINT_MAX 0xffffffff
- #define INTMAX_C(c) c ## L
- #define UINTMAX_C(c) c ## UL

Typedefs

- typedef signed char int8_t
- typedef short int int16_t
- typedef long int int32_t
- typedef unsigned char uint8_t
- typedef unsigned short int uint16_t
- typedef unsigned long int uint32_t
- typedef signed char int_least8_t
- typedef short int int_least16_t
- typedef long int int_least32_t
- typedef unsigned char uint_least8_t
- typedef unsigned short int uint_least16_t
- typedef unsigned long int uint_least32_t
- typedef signed char int_fast8_t
- typedef int int_fast16_t
- typedef long int int_fast32_t
- typedef unsigned char uint_fast8_t
- typedef unsigned int uint_fast16_t
- typedef unsigned long int uint_fast32_t
- typedef int intptr_t
- typedef unsigned int uintptr_t
- typedef long int intmax_t
- typedef unsigned long int uintmax_t

18.42.1 Macro Definition Documentation

18.42.1.1 INT8_MIN `#define INT8_MIN (-128)`

18.42.1.2 INT16_MIN `#define INT16_MIN (-32767-1)`

18.42.1.3 INT32_MIN `#define INT32_MIN (-2147483647L-1)`

18.42.1.4 INT8_MAX `#define INT8_MAX (127)`

18.42.1.5 INT16_MAX `#define INT16_MAX (32767)`

18.42.1.6 INT32_MAX `#define INT32_MAX (2147483647L)`

18.42.1.7 UINT8_MAX `#define UINT8_MAX (255)`

18.42.1.8 UINT16_MAX `#define UINT16_MAX (65535)`

18.42.1.9 UINT32_MAX `#define UINT32_MAX (4294967295UL)`

18.42.1.10 INT_LEAST8_MIN

```
#define INT_LEAST8_MIN INT8_MIN
```

18.42.1.11 INT_LEAST16_MIN

```
#define INT_LEAST16_MIN INT16_MIN
```

18.42.1.12 INT_LEAST32_MIN

```
#define INT_LEAST32_MIN INT32_MIN
```

18.42.1.13 INT_LEAST8_MAX

```
#define INT_LEAST8_MAX INT8_MAX
```

18.42.1.14 INT_LEAST16_MAX

```
#define INT_LEAST16_MAX INT16_MAX
```

18.42.1.15 INT_LEAST32_MAX

```
#define INT_LEAST32_MAX INT32_MAX
```

18.42.1.16 UINT_LEAST8_MAX

```
#define UINT_LEAST8_MAX UINT8_MAX
```

18.42.1.17 UINT_LEAST16_MAX

```
#define UINT_LEAST16_MAX UINT16_MAX
```

18.42.1.18 UINT_LEAST32_MAX

```
#define UINT_LEAST32_MAX UINT32_MAX
```

18.42.1.19 INT_FAST8_MIN

```
#define INT_FAST8_MIN INT8_MIN
```

18.42.1.20 INT_FAST16_MIN

```
#define INT_FAST16_MIN INT16_MIN
```

18.42.1.21 INT_FAST32_MIN

```
#define INT_FAST32_MIN INT32_MIN
```

18.42.1.22 INT_FAST8_MAX

```
#define INT_FAST8_MAX INT8_MAX
```

18.42.1.23 INT_FAST16_MAX

```
#define INT_FAST16_MAX INT16_MAX
```

18.42.1.24 INT_FAST32_MAX

```
#define INT_FAST32_MAX INT32_MAX
```

18.42.1.25 UINT_FAST8_MAX

```
#define UINT_FAST8_MAX UINT8_MAX
```

18.42.1.26 UINT_FAST16_MAX

```
#define UINT_FAST16_MAX UINT16_MAX
```

18.42.1.27 UINT_FAST32_MAX

```
#define UINT_FAST32_MAX UINT32_MAX
```

18.42.1.28 INTPTR_MIN

```
#define INTPTR_MIN (-32767-1)
```

18.42.1.29 INTPTR_MAX

```
#define INTPTR_MAX (32767)
```

18.42.1.30 UINTPTR_MAX

```
#define UINTPTR_MAX (65535)
```

18.42.1.31 INTMAX_MIN

```
#define INTMAX_MIN (-2147483647L-1)
```

18.42.1.32 INTMAX_MAX

```
#define INTMAX_MAX (2147483647L)
```

18.42.1.33 UINTMAX_MAX

```
#define UINTMAX_MAX (4294967295UL)
```

18.42.1.34 PTRDIFF_MIN

```
#define PTRDIFF_MIN (-32767-1)
```

18.42.1.35 PTRDIFF_MAX

```
#define PTRDIFF_MAX (32767)
```

18.42.1.36 SIG_ATOMIC_MIN

```
#define SIG_ATOMIC_MIN (0)
```

18.42.1.37 SIG_ATOMIC_MAX

```
#define SIG_ATOMIC_MAX (255)
```

18.42.1.38 SIZE_MAX

```
#define SIZE_MAX (65535u)
```

18.42.1.39 INT8_C

```
#define INT8_C(
    c ) c
```

18.42.1.40 INT16_C

```
#define INT16_C(
    c ) c
```

18.42.1.41 INT32_C

```
#define INT32_C(
    c ) c ## L
```

18.42.1.42 UINT8_C

```
#define UINT8_C(
    c ) c ## U
```

18.42.1.43 UINT16_C

```
#define UINT16_C(
    c ) c ## U
```

18.42.1.44 UINT32_C #define UINT32_C(c) c ## UL

18.42.1.45 WCHAR_MIN #define WCHAR_MIN 0

18.42.1.46 WCHAR_MAX #define WCHAR_MAX 0xffffffff

18.42.1.47 WINT_MIN #define WINT_MIN 0

18.42.1.48 WINT_MAX #define WINT_MAX 0xffffffff

18.42.1.49 INTMAX_C #define INTMAX_C(c) c ## L

18.42.1.50 UINTMAX_C #define UINTMAX_C(c) c ## UL

18.42.2 Typedef Documentation

18.42.2.1 int8_t typedef signed char int8_t

18.42.2.2 int16_t typedef short int int16_t

18.42.2.3 int32_t typedef long int int32_t

18.42.2.4 uint8_t typedef unsigned char uint8_t

18.42.2.5 uint16_t typedef unsigned short int uint16_t

18.42.2.6 uint32_t typedef unsigned long int uint32_t

18.42.2.7 int_least8_t typedef signed char int_least8_t

18.42.2.8 int_least16_t typedef short int int_least16_t

18.42.2.9 int_least32_t typedef long int int_least32_t

18.42.2.10 uint_least8_t typedef unsigned char uint_least8_t

18.42.2.11 uint_least16_t typedef unsigned short int uint_least16_t

18.42.2.12 uint_least32_t typedef unsigned long int uint_least32_t

18.42.2.13 int_fast8_t typedef signed char int_fast8_t

18.42.2.14 int_fast16_t typedef int int_fast16_t

18.42.2.15 int_fast32_t typedef long int int_fast32_t

18.42.2.16 uint_fast8_t typedef unsigned char uint_fast8_t

18.42.2.17 uint_fast16_t typedef unsigned int uint_fast16_t

18.42.2.18 uint_fast32_t typedef unsigned long int uint_fast32_t

18.42.2.19 intptr_t typedef int intptr_t

18.42.2.20 uintptr_t typedef unsigned int uintptr_t

18.42.2.21 intmax_t typedef long int intmax_t

18.42.2.22 uintmax_t typedef unsigned long int uintmax_t

18.43 stdio.h File Reference

```
#include <types.h>
```

Functions

- void putchar (char c)
- void printf (const char *format,...) NONBANKED
- void sprintf (char *str, const char *format,...) NONBANKED
- void puts (const char *s) NONBANKED
- char * gets (char *s)
- char getchar (void)

18.43.1 Detailed Description

Basic file/console input output functions.
Including stdio.h will use a large number of the background tiles for font characters. If stdio.h is not included then that space will be available for use with other tiles instead.

18.43.2 Function Documentation

18.43.2.1 putchar() `void putchar (`

```
char c )
```

Write the character **c** to stdout.

18.43.2.2 printf() `void printf (`

```
const char * format,
 ...  )
```

Print the string and arguments given by format to stdout.

Parameters

format	The format string as per printf

Does not return the number of characters printed.
Currently supported:

- %hx (char as hex)
- %hu (unsigned char)
- %hd (signed char)
- %c (character)
- %u (unsigned int)
- %d (signed int)
- %x (unsigned int as hex)
- %s (string)

Warning: to correctly pass chars for printing as chars, they *must* be explicitly re-cast as such when calling the function. See docs_chars_varargs for more details.

18.43.2.3 sprintf() `void sprintf (`

```
char * str,
const char * format,
 ...  )
```

Print the string and arguments given by format to a buffer.

Parameters

str	The buffer to print into
format	The format string as per printf

Does not return the number of characters printed.

18.43.2.4 puts() `void puts (`

```
const char * s )
```

puts() writes the string **s** and a trailing newline to stdout.

18.43.2.5 gets() `char* gets ( char * s )`

gets() Reads a line from stdin into a buffer pointed to by **s**.

Parameters

s	Buffer to store string in

Reads until either a terminating newline or an EOF, which it replaces with '\0'. No check for buffer overrun is performed.
Returns: Buffer pointed to by **s**

18.43.2.6 getchar() `char getchar ( void )`

getchar() Reads and returns a single character from stdin.

18.44 stdlib.h File Reference

`#include <types.h>`

Macros

- #define __reentrant

Functions

- void exit (int status) NONBANKED
- int abs (int i)
- long labs (long num)
- int atoi (const char *s)
- long atol (const char *s)
- char * itoa (int n, char *s)
- char * utoa (unsigned int n, char *s)
- char * ltoa (long n, char *s)
- char * ultoa (unsigned long n, char *s)
- void * calloc (size_t nmemb, size_t size)
- void * malloc (size_t size)
- void * realloc (void *ptr, size_t size)
- void free (void *ptr)
- void * bsearch (const void *key, const void *base, size_t nmemb, size_t size, int(*compar)(const void *, const void *) __reentrant)
- void qsort (void *base, size_t nmemb, size_t size, int(*compar)(const void *, const void *) __reentrant)

18.44.1 Macro Definition Documentation

18.44.1.1 __reentrant `#define __reentrant`

file stdlib.h 'Standard library' functions, for whatever that means.

18.44.2 Function Documentation

18.44.2.1 exit() `void exit (`

`int status )`

Causes normal program termination and the value of status is returned to the parent. All open streams are flushed and closed.

18.44.2.2 abs() `int abs (`

`int i )`

Returns the absolute value of int **i**

Parameters

i	Int to obtain absolute value of

If i is negative, returns -i; else returns i.

18.44.2.3 labs() `long labs (`

`long num )`

Returns the absolute value of long int **num**

Parameters

num	Long integer to obtain absolute value of

18.44.2.4 atoi() `int atoi (`

`const char * s )`

Converts an ASCII string to an int

Parameters

s	String to convert to an int

The string may be of the format
`[\s]*[+-][\d]+[\D]*`
i.e. any number of spaces, an optional + or -, then an arbitrary number of digits.
The result is undefined if the number doesnt fit in an int.
Returns: Int value of string

18.44.2.5 atol() `long atol (`

`const char * s )`

Converts an ASCII string to a long.

Parameters

s	String to convert to an long int

See also

atoi()

Returns: Long int value of string

18.44.2.6 itoa() `char* itoa (`

`int n,`
`char * s )`

Converts an int into a base 10 ASCII string.

Parameters

n	Int to convert to a string
s	String to store the converted number

Returns: Pointer to converted string

18.44.2.7 utoa()

```
char* utoa (
    unsigned int n,
    char * s )
```

Converts an unsigned int into a base 10 ASCII string.

Parameters

n	Unsigned Int to convert to a string
s	String to store the converted number

Returns: Pointer to converted string

18.44.2.8 ltoa()

```
char* ltoa (
    long n,
    char * s )
```

Converts a long into a base 10 ASCII string.

Parameters

n	Long int to convert to a string
s	String to store the converted number

Returns: Pointer to converted string

18.44.2.9 ultoa()

```
char* ultoa (
    unsigned long n,
    char * s )
```

Converts an unsigned long into a base 10 ASCII string.

Parameters

n	Unsigned Long Int to convert to a string
s	String to store the converted number

Returns: Pointer to converted string

18.44.2.10 calloc()

```
void* calloc (
    size_t nmemb,
    size_t size )
```

Memory allocation functions

18.44.2.11 malloc()

```
void* malloc (
    size_t size )
```

18.44.2.12 realloc()

```
void* realloc (
    void * ptr,
    size_t size )
```

18.44.2.13 free() void free (
```
void * ptr )
```

18.44.2.14 bsearch() void* bsearch (
```
const void * key,
const void * base,
size_t nmemb,
size_t size,
int(*)(const void *, const void *) __reentrant compar )
```
search a sorted array of **nmemb** items

Parameters

key	Pointer to object that is the key for the search
base	Pointer to first object in the array to search
nmemb	Number of elements in the array
size	Size in bytes of each element in the array
compar	Function used to compare two elements of the array

Returns: Pointer to array entry that matches the search key. If key is not found, NULL is returned.

18.44.2.15 qsort() void qsort (
```
void * base,
size_t nmemb,
size_t size,
int(*)(const void *, const void *) __reentrant compar )
```
Sort an array of **nmemb** items

Parameters

base	Pointer to first object in the array to sort
nmemb	Number of elements in the array
size	Size in bytes of each element in the array
compar	Function used to compare and sort two elements of the array

18.45 stdnoreturn.h File Reference

Macros

- #define noreturn _Noreturn

18.45.1 Macro Definition Documentation

18.45.1.1 noreturn #define noreturn _Noreturn

18.46 string.h File Reference

```
#include <types.h>
```

Functions

- char * strcpy (char *dest, const char *src) NONBANKED __preserves_regs(b
- int strcmp (const char *s1, const char *s2) NONBANKED __preserves_regs(b
- void * memcpy (void *dest, const void *src, size_t len) NONBANKED __preserves_regs(b
- void * memmove (void *dest, const void *src, size_t n)
- void * memset (void *s, int c, size_t n) NONBANKED __preserves_regs(b
- char * reverse (char *s) __preserves_regs(b
- char * strcat (char *s1, const char *s2) NONBANKED
- int strlen (const char *s) NONBANKED __preserves_regs(b
- char * strncat (char *s1, const char *s2, int n) NONBANKED
- int strncmp (const char *s1, const char *s2, int n) NONBANKED
- char * strncpy (char *s1, const char *s2, int n) NONBANKED

Variables

- char c

18.46.1 Detailed Description

Generic string functions.

18.46.2 Function Documentation

18.46.2.1 strcpy()

```
char* strcpy (
        char * dest,
        const char * src )
```

Copies the string pointed to by **src** (including the terminating '\0' character) to the array pointed to by **dest**. The strings may not overlap, and the destination string dest must be large enough to receive the copy.

Parameters

dest	Array to copy into
src	Array to copy from

Returns

A pointer to dest

18.46.2.2 strcmp()

```
int strcmp (
        const char * s1,
        const char * s2 )
```

Compares strings

Parameters

s1	First string to compare
s2	Second string to compare

Returns:

- $>$ 0 if **s1** $>$ **s2**
- 0 if **s1** == **s2**

- $<$ 0 if **s1** $<$ **s2**

18.46.2.3 memcpy() void* memcpy (

```
void * dest,
const void * src,
size_t len )
```

Copies n bytes from memory area src to memory area dest.
The memory areas may not overlap.

Parameters

dest	Buffer to copy into
src	Buffer to copy from
len	Number of Bytes to copy

18.46.2.4 memmove() void* memmove (

```
void * dest,
const void * src,
size_t n )
```

Copies n bytes from memory area src to memory area dest, areas may overlap

18.46.2.5 memset() void* memset (

```
void * s,
int c,
size_t n )
```

Fills the memory region **s** with **n** bytes using value **c**

Parameters

s	Buffer to fill
c	char value to fill with (truncated from int)
n	Number of bytes to fill

18.46.2.6 reverse() char* reverse (

```
char * s )
```

Reverses the characters in a string

Parameters

s	Pointer to string to reverse.

For example 'abcdefg' will become 'gfedcba'.
Banked as the string must be modifiable.
Returns: Pointer to **s**

18.46.2.7 strcat() char* strcat (

```
char * s1,
const char * s2 )
```

Concatenate Strings. Appends string **s2** to the end of string **s1**

Parameters

s1	String to append onto
s2	String to copy from

For example 'abc' and 'def' will become 'abcdef'.
String **s1** must be large enough to store both **s1** and **s2**.
Returns: Pointer to **s1**

18.46.2.8 strlen()

```
int strlen (
    const char * s )
```

Calculates the length of a string

Parameters

s	String to calculate length of

Returns: Length of string not including the terminating '\0' character.

18.46.2.9 strncat()

```
char* strncat (
    char * s1,
    const char * s2,
    int n )
```

Concatenate at most **n** characters from string **s2** onto the end of **s1**.

Parameters

s1	String to append onto
s2	String to copy from
n	Max number of characters to copy from **s2**

String **s1** must be large enough to store both **s1** and **n** characters of **s2**
Returns: Pointer to **s1**

18.46.2.10 strncmp()

```
int strncmp (
    const char * s1,
    const char * s2,
    int n )
```

Compare strings (at most n characters):

Parameters

s1	First string to compare
s2	Second string to compare
n	Max number of characters to compare

Returns:

- $>$ 0 if **s1** $>$ **s2**
- 0 if **s1** == **s2**
- $<$ 0 if **s1** $<$ **s2**

18.46.2.11 strncpy() char* strncpy (char * s1, const char * s2, int n)

Copy **n** characters from string **s2** to **s1**

Parameters

s1	String to copy into
s2	String to copy from
n	Max number of characters to copy from **s2**

If **s2** is shorter than **n**, the remaining bytes in **s1** are filled with \0.
Warning: If there is no \0 in the first **n** bytes of **s2** then **s1** will not be null terminated.
Returns: Pointer to **s1**

18.46.3 Variable Documentation

18.46.3.1 c int c

18.47 time.h File Reference

```
#include <types.h>
```

Macros

- #define CLOCKS_PER_SEC 60

Typedefs

- typedef UINT16 time_t

Functions

- clock_t clock (void) NONBANKED
- time_t time (time_t *t)

18.47.1 Detailed Description

Sort of ANSI compliant time functions.

18.47.2 Macro Definition Documentation

18.47.2.1 CLOCKS_PER_SEC #define CLOCKS_PER_SEC 60

18.47.3 Typedef Documentation

18.47.3.1 time_t typedef UINT16 time_t

18.47.4 Function Documentation

18.47.4.1 clock() `clock_t clock ( void )`

Returns an approximation of processor time used by the program in Clocks
The value returned is the CPU time (ticks) used so far as a clock_t.
To get the number of seconds used, divide by CLOCKS_PER_SEC.
This is based on sys_time, which will wrap around every ~18 minutes. (unsigned 16 bits = 65535 / 60 / 60 = 18.2)

See also

sys_time, time()

18.47.4.2 time() `time_t time ( time_t * t )`

Converts clock() time to Seconds

Parameters

t	If pointer **t** is not NULL, it's value will be set to the same seconds calculation as returned by the function.

The calculation is clock() / CLOCKS_PER_SEC
Returns: time in seconds

See also

sys_time, clock()

18.48 typeof.h File Reference

Macros

- #define TYPEOF_INT 1
- #define TYPEOF_SHORT 2
- #define TYPEOF_CHAR 3
- #define TYPEOF_LONG 4
- #define TYPEOF_FLOAT 5
- #define TYPEOF_FIXED16X16 6
- #define TYPEOF_BIT 7
- #define TYPEOF_BITFIELD 8
- #define TYPEOF_SBIT 9
- #define TYPEOF_SFR 10
- #define TYPEOF_VOID 11
- #define TYPEOF_STRUCT 12
- #define TYPEOF_ARRAY 13
- #define TYPEOF_FUNCTION 14
- #define TYPEOF_POINTER 15
- #define TYPEOF_FPOINTER 16
- #define TYPEOF_CPOINTER 17
- #define TYPEOF_GPOINTER 18
- #define TYPEOF_PPOINTER 19
- #define TYPEOF_IPOINTER 20
- #define TYPEOF_EEPPOINTER 21

18.48.1 Macro Definition Documentation

18.48.1.1 TYPEOF_INT

```
#define TYPEOF_INT 1
```

18.48.1.2 TYPEOF_SHORT

```
#define TYPEOF_SHORT 2
```

18.48.1.3 TYPEOF_CHAR

```
#define TYPEOF_CHAR 3
```

18.48.1.4 TYPEOF_LONG

```
#define TYPEOF_LONG 4
```

18.48.1.5 TYPEOF_FLOAT

```
#define TYPEOF_FLOAT 5
```

18.48.1.6 TYPEOF_FIXED16X16

```
#define TYPEOF_FIXED16X16 6
```

18.48.1.7 TYPEOF_BIT

```
#define TYPEOF_BIT 7
```

18.48.1.8 TYPEOF_BITFIELD

```
#define TYPEOF_BITFIELD 8
```

18.48.1.9 TYPEOF_SBIT

```
#define TYPEOF_SBIT 9
```

18.48.1.10 TYPEOF_SFR

```
#define TYPEOF_SFR 10
```

18.48.1.11 TYPEOF_VOID

```
#define TYPEOF_VOID 11
```

18.48.1.12 TYPEOF_STRUCT

```
#define TYPEOF_STRUCT 12
```

18.48.1.13 TYPEOF_ARRAY

```
#define TYPEOF_ARRAY 13
```

18.48.1.14 TYPEOF_FUNCTION

```
#define TYPEOF_FUNCTION 14
```

18.48.1.15 TYPEOF_POINTER

```
#define TYPEOF_POINTER 15
```

18.48.1.16 TYPEOF_FPOINTER

```
#define TYPEOF_FPOINTER 16
```

18.48.1.17 TYPEOF_CPOINTER

```
#define TYPEOF_CPOINTER 17
```

18.48.1.18 TYPEOF_GPOINTER

```
#define TYPEOF_GPOINTER 18
```

18.48.1.19 TYPEOF_PPOINTER

```
#define TYPEOF_PPOINTER 19
```

18.48.1.20 TYPEOF_IPOINTER

```
#define TYPEOF_IPOINTER 20
```

18.48.1.21 TYPEOF_EEPPOINTER

```
#define TYPEOF_EEPPOINTER 21
```

Index