wflash - WiFi bootloader for Genesis/Megadrive

Generated by Doxygen 1.8.13

Contents

Chapter 1

Todo List

Module lsd

Implement UART RTS/CTS handshaking.

Current implementation uses polling. Unfortunately as the Genesis/ Megadrive does not have an interrupt pin on the cart, implementing more efficient data transmission techniques will be tricky.

Proper implementation of error handling.

Module MegaWiFi

Missing a lot of integrity checks, also module should track used channels, and is not currently doing it

Module sysfsm

Module currently does not support checksum/CRC

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

Module Documentation

4.1 flash

This module allows to manage (mainly read and write) from flash memory chips such as S29GL032.

Data Structures

- struct FlashCmd

  Data used to perform different flash commands.

Macros

- #define FLASH_CHIP_LENGTH (4LU∗1024LU∗1024LU)

  Flash chip length: 4 MiB.
- #define FLASH_CHIP_WBUFLEN 16

  Write data buffer length in words.
- #define FLASH_RESET_CYC 1

  Number of write cycles to reset Flash interface.
- #define FLASH_UNLOCK_CYC 2

  Number of cycles to unlock.
- #define FLASH_AUTOSEL_CYC 1

  Number of cycles of the autoselect command.
- #define FLASH_MANID_CYC 1

  Number of cycles of the manufacturer ID request command.
- #define FLASH_DEVID_CYC 3

  Number of cycles of the device ID request command.
- #define FLASH_PROG_CYC 1

  Number of cycles of the program command.
- #define FLASH_WR_BUF_CYC 1

  Number of cycles of the write to buffer command.
- #define FLASH_PRG_BUF_CYC 1

  Number of cycles of the program buffer to flash command.
- #define FLASH_UL_BYP_CYC 1

  Number of cycles of the unlock bypass command.

- #define FLASH_UL_BYP_PROG_CYC 1

  Number of cycles of the unlock bypass program command.
- #define FLASH_UL_BYP_RST_CYC 2

  Number of cycles of the unlock bypass reset command.
- #define FLASH_CHIP_ERASE_CYC 4

  Number of cycles of the chip erase command.
- #define FLASH_SEC_ERASE_CYC 3

  Number of cycles of the sector erase command.
- #define FLASH_WRITE_CMD(cmd, iterator)

Functions

- void FlashInit (void)

  Module initialization. Configures the 68k bus.
- void FlashIdle (void)

  Set flash ports to default (idle) values.
- uint8_t FlashGetManId (void)

  Writes the manufacturer ID query command to the flash chip.
- void FlashGetDevId (uint8_t devId[3])

  Writes the device ID query command to the flash chip.
- void FlashProg (uint32_t addr, uint16_t data)

  Programs a word to the specified address.
- uint8_t FlashWriteBuf (uint32_t addr, uint16_t data[ ], uint8_t wLen)

  Programs a buffer to the specified address range.
- void FlashUnlockBypass (void)
- void FlashUnlockBypassReset (void)
- uint8_t FlashChipErase (void)
- uint8_t FlashSectErase (uint32_t addr)
- uint8_t FlashRangeErase (uint32_t addr, uint32_t len)
- uint8_t FlashDataPoll (uint32_t addr, uint8_t data)

  Polls flash chip after a program operation, and returns when the program operation ends, or when there is an error.
- uint8_t FlashErasePoll (uint32_t addr)

  Polls flash chip after an erase operation, and returns when the program operation ends, or when there is an error.

4.1.1 Detailed Description

This module allows to manage (mainly read and write) from flash memory chips such as S29GL032.

Author

Jesús Alonso (doragasu)

Date

2015

4.1.2 Macro Definition Documentation

4.1.2.1 FLASH_WRITE_CMD

```
#define FLASH_WRITE_CMD(
                cmd,
                iterator )
```

Value:

```
do {                    \
    for (iterator = 0; iterator < cmd ## _CYC; iterator++) {    \
        FlashWrite(cmd[iterator].addr, cmd[iterator].data);     \
    }                                                           \
} while (0);
```

Helper macro for writing commands. Takes the command and an auxiliar variable used as an iterator.

Definition at line 204 of file flash.h.

4.1.3 Function Documentation

4.1.3.1 FlashChipErase()

```
uint8_t FlashChipErase (
              void  )
```

Erases the complete flash chip.

Returns

'0' the if erase operation completed successfully, '1' otherwise.

Definition at line 239 of file flash.c.

4.1.3.2 FlashDataPoll()

```
uint8_t FlashDataPoll (
              uint32_t addr,
              uint8_t data )
```

Polls flash chip after a program operation, and returns when the program operation ends, or when there is an error.

Parameters

in	*addr*	Address to which data has been written.
in	*data*	Data written to addr address.

Returns

0 if OK, 1 if error during program operation.

Definition at line 55 of file flash.c.

4.1.3.3 FlashErasePoll()

```
uint8_t FlashErasePoll (
            uint32_t addr )
```

Polls flash chip after an erase operation, and returns when the program operation ends, or when there is an error.

Parameters

in	*addr*	Address contained in the erased zone.

Returns

0 if OK, 1 if error during program operation.

Warning

Function erases the minimum memory range CONTAINING the specified range. Due to the granularity of the flash sectors, it can (and most likely will) erase more memory than requested. This is expected behaviour, and programmer must be aware of this.
Currently the function does not check if the input range covers the sector/s containing the bootloader.

Parameters

in	*addr*	Address contained in the erased zone.

Returns

1 if OK, 0 if error during program operation.

Definition at line 87 of file flash.c.

4.1.3.4 FlashGetDevId()

```
void FlashGetDevId (
            uint8_t devId[3] )
```

Writes the device ID query command to the flash chip.

Parameters

out	*devId*	The device ID code, consisting of 3 words.

Definition at line 135 of file flash.c.

4.1.3.5 FlashGetManId()

```
uint8_t FlashGetManId (
            void  )
```

Writes the manufacturer ID query command to the flash chip.

Returns

The manufacturer ID code.

Definition at line 119 of file flash.c.

4.1.3.6 FlashInit()

```
void FlashInit (
            void  )
```

Module initialization. Configures the 68k bus.

Public functions

Definition at line 111 of file flash.c.

4.1.3.7 FlashProg()

```
void FlashProg (
            uint32_t addr,
            uint16_t data )
```

Programs a word to the specified address.

Parameters

in	*addr*	The address to which data will be programmed.
in	*data*	Data to program to the specified address.

Warning

Doesn't poll until programming is complete

Definition at line 152 of file flash.c.

4.1.3.8 FlashRangeErase()

```
uint8_t FlashRangeErase (
            uint32_t addr,
            uint32_t len )
```

Erases a flash memory range.

Parameters

in	*addr*	Address base for the range to erase.
in	*len*	Length of the range to erase

Returns

'0' if the erase operation completed successfully, '1' otherwise.

Erases a flash memory range.

Parameters

in	*addr*	Address base for the range to erase.
in	*len*	Length of the range to erase

Returns

'0' if the erase operation completed successfully, '1' otherwise.

Warning

Function erases the minimum memory range CONTAINING the specified range. Due to the granularity of the flash sectors, it can (and most likely will) erase more memory than requested. This is expected behaviour, and programmer must be aware of this.
Currently the function does not check if the input range covers the sector/s containing the bootloader.

Definition at line 289 of file flash.c.

4.1.3.9 FlashSectErase()

```
uint8_t FlashSectErase (
            uint32_t addr )
```

Erases a complete flash sector, specified by addr parameter.

Parameters

in	*addr*	Address contained in the sector that will be erased.

Returns

'0' if the erase operation completed successfully, '1' otherwise.

Definition at line 255 of file flash.c.

4.1.3.10 FlashUnlockBypass()

```
void FlashUnlockBypass (
            void  )
```

Enables the "Unlock Bypass" status, allowing to issue several commands (like the Unlock Bypass Programm) using less write cycles.

Definition at line 218 of file flash.c.

4.1.3.11 FlashUnlockBypassReset()

```
void FlashUnlockBypassReset (
            void  )
```

Ends the "Unlock Bypass" state, returning to default read mode.

Definition at line 228 of file flash.c.

4.1.3.12 FlashWriteBuf()

```
uint8_t FlashWriteBuf (
            uint32_t addr,
            uint16_t data[],
            uint8_t wLen )
```

Programs a buffer to the specified address range.

Parameters

in	*addr*	The address of the first word to be written
in	*data*	The data array to program to the specified address range.
in	*wLen*	The number of words to program, contained on data.

Returns

The number of words successfully programed.

Note

wLen must be less or equal than 16.
If addr-wLen defined range crosses a write-buffer boundary, all the requested words will not be written. To avoid this situation, it is advisable to write to addresses having the lower 4 bits (A1∼A5) equal to 0.

Definition at line 176 of file flash.c.

4.2 gamepad

Genesis/Megadrive routines for reading the Gamepad.

Modules

- GpRegAddrs

  Gamepad related register addresses.
- GpRegs

  Gamepad related registers.
- GpMasks

  Masks used to filter the cross and buttons.

Macros

- #define GpInit() do{GP_REG_PORTA_CTRL = 0x40;}while(0)

Functions

- uint8_t GpRead (void)

  Genesis/Megadrive routines for reading the Gamepad.

4.2.1 Detailed Description

Genesis/Megadrive routines for reading the Gamepad.

Author

Jesús Alonso (doragasu)

Date

2017

Note

Currently only 3-button gamepad for port A is supported. The routine is also compatible with 6-button pads as long as the GpRead() routine is called only once per frame.

4.2.2 Macro Definition Documentation

4.2.2.1 GpInit

```
#define GpInit( ) do{GP_REG_PORTA_CTRL = 0x40;}while(0)
```

Module initialization. Call this routine before using any other in this module.

Definition at line 78 of file gamepad.h.

4.2.3 Function Documentation

4.2.3.1 GpRead()

```
uint8_t GpRead (
            void  )
```

Genesis/Megadrive routines for reading the Gamepad.

Read gamepad. Currently only 3-button pads on port A are read.

Returns

Pad status in format SACBRLDU (START, A, C, B, RIGHT, LEFT, DOWN, UP). For each bit, a '1' means that the button/direction is not pressed. A '0' means that the button/direction is pressed. Masks can be used to filter returned data (see GpMasks).

Author

Jesús Alonso (doragasu)

Date

2017

Note

Currently only 3-button gamepad for port A is supported. The routine is also compatible with 6-button pads as long as the GpRead() routine is called only once per frame. Read gamepad. Currently only 3-button pads on port A are read.

Returns

Pad status in format SACBRLDU (START, A, C, B, RIGHT, LEFT, DOWN, UP). For each bit, a '1' means that the button/direction is not pressed. A '0' means that the button/direction is pressed. Masks can be used to filter returned data (see GpMasks).

Definition at line 20 of file gamepad.c.

4.3 main

Wireless Flash memory manager for MegaWiFi cartridges. Allows remotely programming MegaWiFi catridges. It also allows performing some flash memory management routienes, like erasing the cartridge, reading data, reading flash identifiers, etc.

Macros

- #define WFLASH_BUFLEN WF_MAX_DATALEN

  Length of the wflash buffer.
- #define WFLASH_PORT 1989

  TCP port to use (set to Megadrive release year ;-)

Functions

- void Panic (char msg[ ])

  Sets background to RED, prints message and loops forever.
- int WaitUserInteraction (void)

  Waits until start is pressed (to boot flashed ROM) or a client connects.
- int WaitApJoin (void)

  Waits forever until module joins an AP or error.
- int MegaWifiInit (void)

  MegaWiFi initialization.
- void Init (void)

  Global initialization.
- int main (void)

  Entry point.

4.3.1 Detailed Description

Wireless Flash memory manager for MegaWiFi cartridges. Allows remotely programming MegaWiFi catridges. It also allows performing some flash memory management routienes, like erasing the cartridge, reading data, reading flash identifiers, etc.

Author

Jesús Alonso (doragasu)

Date

2017

4.3.2 Function Documentation

4.3.2.1 main()

```
int main (
            void  )
```

Entry point.

Error or socket disconnected

Definition at line 128 of file main.c.

4.4 16c550

Simple 16C550 UART chip driver.

Modules

- UartRegs

  16C550 UART registers
- UartOuts

  Output pins controlled by the MCR UART register.
- UartIns

  Input pins readed in the MSR UART register.

Data Structures

- struct UartShadow

  Structure with the shadow registers.

Macros

- #define UART_BASE 0xA130C1

  16C550 UART base address
- #define UART_CLK 24000000LU

  Clock applied to 16C550 chip. Currently using 24 MHz crystal.
- #define UART_BR 1500000LU
- #define UART_TX_FIFO_LEN 16

  Length of the TX FIFO in bytes.
- #define DivWithRounding(dividend, divisor) ((((dividend)∗2/(divisor))+1)/2)

  Division with one bit rounding, useful for divisor calculations.
- #define UART_DLM_VAL (DivWithRounding(UART_CLK, 16 ∗ UART_BR)>>8)

  Value to load on the UART divisor, high byte.
- #define UART_DLL_VAL (DivWithRounding(UART_CLK, 16 ∗ UART_BR) & 0xFF)

  Value to load on the UART divisor, low byte.
- #define UartTxReady() (UART_LSR & 0x20)

  Checks if UART transmit register/FIFO is ready. In FIFO mode, up to 16 characters can be loaded each time transmitter is ready.
- #define UartRxReady() (UART_LSR & 0x01)

  Checks if UART receive register/FIFO has data available.
- #define UartPutc(c) do{UART_RHR = (c);}while(0);

  Sends a character. Please make sure there is room in the transmit register/FIFO by calling UartRxReady() before using this function.
- #define UartGetc() (UART_RHR)

  Returns a received character. Please make sure data is available by calling UartRxReady() before using this function.
- #define UartSet(reg, val) do{sh.reg = (val);UART_##reg = (val);}while(0)

  Sets a value in IER, FCR, LCR or MCR register.
- #define UartGet(reg) (sh.reg)

  Gets value of IER, FCR, LCR or MCR register.
- #define UartSetBits(reg, val)

  Sets bits in IER, FCR, LCR or MCR register.
- #define UartClrBits(reg, val)

  Clears bits in IER, FCR, LCR or MCR register.
- #define UartResetFifos() UartSetBits(FCR, 0x07)

  Reset TX and RX FIFOs.

Functions

- void UartInit (void)

  Initializes the driver. The baud rate is set to UART_BR, and the UART FIFOs are enabled. This function must be called before using any other API call.

Variables

- UartShadow sh

  Uart shadow registers. Do NOT access directly!

4.4.1 Detailed Description

Simple 16C550 UART chip driver.

Author

Jesus Alonso (doragasu)

Date

2016

4.4.2 Macro Definition Documentation

4.4.2.1 UART_BR

```
#define UART_BR 1500000LU
```

Desired baud rate. Maximum achievable baudrate with 24 MHz crystal is 24000000/16 = 1.5 Mbps

Definition at line 24 of file 16c550.h.

4.4.2.2 UartClrBits

```
#define UartClrBits(
              reg,
              val )
```

Value:

```
do{sh.reg &= ~(val);                          \
                                UART_##reg = sh.reg;}while(0)
```

Clears bits in IER, FCR, LCR or MCR register.

Parameters

in	*reg*	Register to modify (IER, FCR, LCR or MCR).
in	*val*	Bits set in val, will be cleared in reg register.

Definition at line 167 of file 16c550.h.

4.4.2.3 UartGet

```
#define UartGet(
              reg ) (sh.reg)
```

Gets value of IER, FCR, LCR or MCR register.

Parameters

in	*reg*	Register to read (IER, FCR, LCR or MCR).

Returns

The value of the requested register.

Definition at line 150 of file 16c550.h.

4.4.2.4 UartGetc

```
#define UartGetc( ) (UART_RHR)
```

Returns a received character. Please make sure data is available by calling UartRxReady() before using this function.

Returns

Received character.

Definition at line 134 of file 16c550.h.

4.4.2.5 UartPutc

```
#define UartPutc(
              c ) do{UART_RHR = (c);}while(0);
```

Sends a character. Please make sure there is room in the transmit register/FIFO by calling UartRxReady() before using this function.

Returns

Received character.

Definition at line 126 of file 16c550.h.

4.4.2.6 UartRxReady

```
#define UartRxReady( ) (UART_LSR & 0x01)
```

Checks if UART receive register/FIFO has data available.

Returns

TRUE if at least 1 byte is available, FALSE otherwise.

Definition at line 118 of file 16c550.h.

4.4.2.7 UartSet

```
#define UartSet(
              reg,
              val ) do{sh.reg = (val);UART_##reg = (val);}while(0)
```

Sets a value in IER, FCR, LCR or MCR register.

Parameters

in	*reg*	Register to modify (IER, FCR, LCR or MCR).
in	*val*	Value to set in IER, FCR, LCR or MCR register.

Definition at line 142 of file 16c550.h.

4.4.2.8 UartSetBits

```
#define UartSetBits(
              reg,
              val )
```

Value:

```
do{sh.reg |= (val);                                \
                                  UART_##reg = sh.reg;}while(0)
```

Sets bits in IER, FCR, LCR or MCR register.

Parameters

in	*reg*	Register to modify (IER, FCR, LCR or MCR).
in	*val*	Bits set in val, will be set in reg register.

Definition at line 158 of file 16c550.h.

4.4.2.9 UartTxReady

```
#define UartTxReady( ) (UART_LSR & 0x20)
```

Checks if UART transmit register/FIFO is ready. In FIFO mode, up to 16 characters can be loaded each time transmitter is ready.

Returns

TRUE if transmitter is ready, FALSE otherwise.

Definition at line 111 of file 16c550.h.

4.5 lsd

Local Symmetric Data-link. Implements an extremely simple protocol to link two full-duplex devices, multiplexing the data link.

Modules

- ReturnCodes

  OK/Error codes returned by several functions.

Macros

- #define LSD_OVERHEAD 4

  LSD frame overhead in bytes.
- #define LSD_UART 0

  Uart used for LSD.
- #define LSD_STX_ETX 0x7E

  Start/end of transmission character.
- #define LSD_MAX_CH 4

  Maximum number of available simultaneous channels.
- #define LSD_RECV_PRIO 2

  Receive task priority.
- #define LSD_MAX_LEN 4095

  Maximum data payload length.

Functions

- void LsdInit (void)
- int LsdChEnable (uint8_t ch)
- int LsdChDisable (uint8_t ch)
- int LsdSend (uint8_t *data, uint16_t len, uint8_t ch, uint32_t maxLoopCnt)
- int LsdSplitStart (uint8_t *data, uint16_t len, uint16_t total, uint8_t ch, uint32_t maxLoopCnt)
- int LsdSplitNext (uint8_t *data, uint16_t len, uint32_t maxLoopCnt)
- int LsdSplitEnd (uint8_t *data, uint16_t len, uint32_t maxLoopCnt)
- int LsdRecv (uint8_t *buf, uint16_t *maxLen, uint32_t maxLoopCnt)

4.5.1 Detailed Description

Local Symmetric Data-link. Implements an extremely simple protocol to link two full-duplex devices, multiplexing the data link.

Author

Jesus Alonso (doragasu)

Date

2016

Todo Implement UART RTS/CTS handshaking.

Current implementation uses polling. Unfortunately as the Genesis/ Megadrive does not have an interrupt pin on the cart, implementing more efficient data transmission techniques will be tricky.

Proper implementation of error handling.

4.5.2 Function Documentation

4.5.2.1 LsdChDisable()

```
int LsdChDisable (
            uint8_t ch )
```

Disables a channel to stop reception and prohibit sending data.

Parameters

in	*ch*	Channel number.

Returns

A pointer to an empty TX buffer, or NULL if no buffer is available.

Definition at line 104 of file lsd.c.

4.5.2.2 LsdChEnable()

```
int LsdChEnable (
            uint8_t ch )
```

Enables a channel to start reception and be able to send data.

Parameters

in	*ch*	Channel number.

Returns

A pointer to an empty TX buffer, or NULL if no buffer is available.

Definition at line 89 of file lsd.c.

4.5.2.3 LsdInit()

```
void LsdInit (
            void  )
```

Module initialization. Call this function before any other one in this module.

Definition at line 70 of file lsd.c.

4.5.2.4 LsdRecv()

```
int LsdRecv (
            uint8_t * buf,
            uint16_t * maxLen,
            uint32_t maxLoopCnt )
```

Receives a frame using LSD protocol.

Parameters

out	*buf*	Buffer that will hold the received data.
in,out	*maxLen*	When calling the function, the variable pointed by maxLen, must hold the maximum number of bytes buf can store. On return, the variable is updated to the number of bytes received.
in	*maxLoopCnt*	Maximum number of loops trying to read data.

Returns

On success, the number of the channel in which data has been received. On failure, a negative number.

Definition at line 256 of file lsd.c.

4.5.2.5 LsdSend()

```
int LsdSend (
            uint8_t * data,
            uint16_t len,
            uint8_t ch,
            uint32_t maxLoopCnt )
```

Sends data through a previously enabled channel.

Parameters

in	*data*	Buffer to send.
in	*len*	Length of the buffer to send.
in	*ch*	Channel number to use.
in	*maxLoopCnt*	Maximum number of loops trying to write data.

Returns

-1 if there was an error, or the number of characterse sent otherwise. Note returned value might be 0 if no characters were sent due to maxLoopCnt value reached (timeout).

Note

maxLoopCnt value is only used for the wait before starting sending the frame header. For sending the data payload and the ETX, UINT32_MAX value is used for loop counts. If tighter control of the timing is necessary, frame must be sent using split functions.

Definition at line 131 of file lsd.c.

4.5.2.6 LsdSplitEnd()

```
int LsdSplitEnd (
            uint8_t * data,
            uint16_t len,
            uint32_t maxLoopCnt )
```

Appends (sends) additional data to a frame previously started by an LsdSplitStart() call, and finally ends the frame.

Parameters

in	*data*	Buffer to send.
in	*len*	Length of the data buffer to send.
in	*maxLoopCnt*	Maximum number of loops trying to write data.

Returns

-1 if there was an error, or the number of characterse sent otherwise.

Definition at line 227 of file lsd.c.

4.5.2.7 LsdSplitNext()

```
int LsdSplitNext (
            uint8_t * data,
            uint16_t len,
            uint32_t maxLoopCnt )
```

Appends (sends) additional data to a frame previously started by an LsdSplitStart() call.

Parameters

in	*data*	Buffer to send.
in	*len*	Length of the data buffer to send.
in	*maxLoopCnt*	Maximum number of loops trying to write data.

Returns

-1 if there was an error, or the number of characterse sent otherwise.

Definition at line 210 of file lsd.c.

4.5.2.8 LsdSplitStart()

```
int LsdSplitStart (
            uint8_t * data,
            uint16_t len,
            uint16_t total,
            uint8_t ch,
            uint32_t maxLoopCnt )
```

Starts sending data through a previously enabled channel. Once started, you can send more additional data inside of the frame by issuing as many LsdSplitNext() calls as needed, and end the frame by calling LsdSplitEnd().

Parameters

in	*data*	Buffer to send.
in	*len*	Length of the data buffer to send.
in	*total*	Total length of the data to send using a split frame.
in	*ch*	Channel number to use for sending.
in	*maxLoopCnt*	Maximum number of loops trying to write data.

Returns

-1 if there was an error, or the number of characterse sent otherwise.

Note

maxLoopCnt is only used for the wait before starting sending the frame header. Optional data field is sent using UINT32_MAX as loop count.

Definition at line 180 of file lsd.c.

4.6 megawifi

MeGaWiFi API implementation.

Modules

- MwCtrlPins

  Pins used to control WiFi module.
- MwRetVals

  Function return values.

Data Structures

- struct MwApData

  Access Point data.

Macros

- #define MW_SSID_MAXLEN 32

  Maximum SSID length (including '\0').
- #define MW_PASS_MAXLEN 64

  Maximum password length (including '\0').
- #define MW_NTP_POOL_MAXLEN 80

  Maximum length of an NTP pool URI (including '\0').
- #define MW_NUM_AP_CFGS 3

  Number of AP configurations stored to nvflash.
- #define MW_NUM_DNS_SERVERS 2

  Number of DSN servers supported per AP configuration.
- #define MW_FSM_QUEUE_LEN 8

  Length of the FSM queue.
- #define MW_MAX_SOCK 3

  Maximum number of simultaneous TCP connections.
- #define MW_CTRL_CH 0

  Control channel used for LSD protocol.
- #define MW_DEF_MAX_LOOP_CNT UINT32_MAX
- #define MW_CMD_MIN_BUFLEN 104
- #define MwSend(ch, data, length)

  Sends data through a socket, using a previously allocated channel.
- #define MwModuleReset() do{UartSetBits(MCR, MW__RESET);}while(0)

  Puts the WiFi module in reset state.
- #define MwModuleStart() do{UartClrBits(MCR, MW__RESET);}while(0)

  Releases the module from reset state.

Functions

- int MwInit (char *cmdBuf, uint16_t bufLen)

  MwInit Module initialization. Must be called once before using any other function. It also initializes de UART.
- int MwVersionGet (uint8_t *verMajor, uint8_t *verMinor, char *variant[ ])

  Obtain module version numbers and string.
- int MwDefaultCfgSet (void)

  Set default module configuration.
- int MwApCfgSet (uint8_t index, const char ssid[ ], const char pass[ ])

  Set access point configuration (SSID and password).
- int MwApCfgGet (uint8_t index, char *ssid[ ], char *pass[ ])

  Gets access point configuration (SSID and password).
- int MwIpCfgSet (uint8_t index, const MwIpCfg *ip)

  Set IPv4 configuration.
- int MwIpCfgGet (uint8_t index, MwIpCfg **ip)

  Get IPv4 configuration.
- int MwApScan (char *apData[ ])

  Scan for access points.
- int MwApFillNext (char apData[ ], uint16_t pos, MwApData *apd, uint16_t dataLen)

  Parses received AP data and fills information of the AP at "pos". Useful to extract AP information from the data obtained by calling MwApScan() function.
- int MwApJoin (uint8_t index)

  Tries joining an AP. If successful, also configures IPv4.
- int MwApLeave (void)

  Leaves a previously joined AP.
- int MwTcpConnect (uint8_t ch, char dstaddr[ ], char dstport[ ], char srcport[ ])

  Tries establishing a TCP connection with specified server.
- int MwTcpDisconnect (uint8_t ch)

  Disconnects a TCP socket from specified channel.
- int MwTcpBind (uint8_t ch, uint16_t port)

  Binds a socket to a port, and listens to connections on the port. If a connection request is received, it will be automatically accepted.
- int MwDataWait (uint32_t maxLoopCnt)

  Waits until data is received or loop timeout. If data is received, return the channel on which it has been.
- int MwRecv (uint8_t **data, uint16_t *len, uint32_t maxLoopCnt)

  Receive data.
- MwMsgSysStat * MwSysStatGet (void)

  Get system status.
- MwSockStat MwSockStatGet (uint8_t ch)

  Get socket status.
- int MwSntpCfgSet (char *servers[3], uint8_t upDelay, char timezone, char dst)

  Configure SNTP parameters and timezone.
- char * MwDatetimeGet (uint32_t dtBin[2])

  Get date and time.
- int MwFlashSectorErase (uint16_t sect)

  Erase a 4 KiB Flash sector. Every byte of an erased sector can be read as 0xFF.
- int MwFlashWrite (uint32_t addr, uint8_t data[ ], uint16_t dataLen)

  Write data to specified flash address.
- uint8_t * MwFlashRead (uint32_t addr, uint16_t dataLen)

  Read data from specified flash address.
- int MwCmdSend (MwCmd *cmd, uint32_t maxLoopCnt)

  Send a command to the WiFi module.
- int MwCmdReplyGet (MwCmd *rep, uint32_t maxLoopCnt)

  Try obtaining a reply to a command.

4.6.1 Detailed Description

MeGaWiFi API implementation.

Author

Jesus Alonso (doragasu)

Date

2015

Note

Module is not reentrant.

Todo Missing a lot of integrity checks, also module should track used channels, and is not currently doing it

4.6.2 Macro Definition Documentation

4.6.2.1 MW_CMD_MIN_BUFLEN

```
#define MW_CMD_MIN_BUFLEN 104
```

Minimum command buffer length to be able to send all available commands with minimum data payload. This length might not guarantee that commands like MwSntpCfgSet() can be sent if payload length is big enough).

Definition at line 64 of file megawifi.h.

4.6.2.2 MW_DEF_MAX_LOOP_CNT

```
#define MW_DEF_MAX_LOOP_CNT UINT32_MAX
```

Default value of maximum times to try completing a command before desisting

Definition at line 59 of file megawifi.h.

4.6.2.3 MwSend

```
#define MwSend(
                ch,
                data,
                length )
```

Value:

```
LsdSend(data, length, ch, \
                                        MW_DEF_MAX_LOOP_CNT)
```

Sends data through a socket, using a previously allocated channel.

Parameters

in	*ch*	Channel used to send the data.
in	*data*	Data to send through channel.
in	*length*	Length in bytes of the data field.

Definition at line 276 of file megawifi.h.

4.6.3 Function Documentation

4.6.3.1 MwApCfgGet()

```
int MwApCfgGet (
            uint8_t index,
            char * ssid[],
            char * pass[] )
```

Gets access point configuration (SSID and password).

Parameters

in	*index*	Index of the configuration to get.
out	*ssid*	String with the AP SSID got.
out	*pass*	String with the AP SSID got.

Returns

MW_OK if configuration successfully got, MW_ERROR otherwise.

Warning

ssid is zero padded up to 32 bytes, and pass is zero padded up to 64 bytes. If ssid is 32 bytes, it will NOT be NULL terminated. Also if pass is 64 bytes, it will NOT be NULL terminated.

Definition at line 232 of file megawifi.c.

4.6.3.2 MwApCfgSet()

```
int MwApCfgSet (
            uint8_t index,
            const char ssid[],
            const char pass[] )
```

Set access point configuration (SSID and password).

Parameters

in	*index*	Index of the configuration to set.
in	*ssid*	String with the AP SSID to set.
in	*pass*	String with the AP SSID to set.

Returns

MW_OK if configuration successfully set, MW_ERROR otherwise.

Note

Strings must be NULL terminated. Maximum SSID length is 32 bytes, maximum pass length is 64 bytes.

Definition at line 197 of file megawifi.c.

4.6.3.3 MwApFillNext()

```
int MwApFillNext (
            char apData[],
            uint16_t pos,
            MwApData * apd,
            uint16_t dataLen )
```

Parses received AP data and fills information of the AP at "pos". Useful to extract AP information from the data obtained by calling MwApScan() function.

Parameters

in	*apData*	Access point data obtained from MwApScan().
in	*pos*	Position at which to extract data.
out	*apd*	Pointer to the extracted data from an AP.
in	*dataLen*	Lenght of apData.

Returns

Position of the next AP entry in apData, 0 if no more APs available or MW_ERROR if apData/pos combination is not valid.

Note

This functions executes locally, does not communicate with the WiFi module.

Definition at line 332 of file megawifi.c.

4.6.3.4 MwApJoin()

```
int MwApJoin (
            uint8_t index )
```

Tries joining an AP. If successful, also configures IPv4.

Parameters

in	*index*	Index of the configuration used to join the AP.

Returns

MW_OK if AP joined successfully and ready to send/receive data, or MW_ERROR if AP join/IP configuration failed.

Definition at line 356 of file megawifi.c.

4.6.3.5 MwApLeave()

```
int MwApLeave (
            void  )
```

Leaves a previously joined AP.

Returns

MW_OK if AP successfully left, or MW_ERROR if operation failed.

Definition at line 373 of file megawifi.c.

4.6.3.6 MwApScan()

```
int MwApScan (
            char * apData[] )
```

Scan for access points.

Parameters

out	*apData*	Data of the found access points. Each entry has the format specified on the MwApData structure.

Returns

Length in bytes of the output data if operation completes successfully, or MW_ERROR if scan fails.

Definition at line 304 of file megawifi.c.

4.6.3.7 MwCmdReplyGet()

```
int MwCmdReplyGet (
            MwCmd * rep,
            uint32_t maxLoopCnt )
```

Try obtaining a reply to a command.

Parameters

out	*rep*	Pointer to MwRep structure, containing the reply to the command, if the call completed successfully.
in	*maxLoopCnt*	Maximum number of loops trying to read data.

Returns

The channel on which the data has been received (0 if it was on the control channel). Lower than 0 if there was a reception error.

Definition at line 101 of file megawifi.c.

4.6.3.8 MwCmdSend()

```
int MwCmdSend (
            MwCmd * cmd,
            uint32_t maxLoopCnt )
```

Send a command to the WiFi module.

Parameters

in	*cmd*	Pointer to the filled MwCmd command structure.
in	*maxLoopCnt*	Maximum number of loops trying to write command.

Returns

0 if OK. Nonzero if error.

Definition at line 82 of file megawifi.c.

4.6.3.9 MwDataWait()

```
int MwDataWait (
            uint32_t maxLoopCnt )
```

Waits until data is received or loop timeout. If data is received, return the channel on which it has been.

Parameters

in	*maxLoopCnt*	Maximum number of loop tries before desisting from waiting. Set to 0 avoid waiting if no data is available.

Returns

Channel in which data has been received, or MW_ERROR if an error has occurred.

Note

If data has been received on control channel, 0 will be returned.

Definition at line 486 of file megawifi.c.

4.6.3.10 MwDatetimeGet()

```
char* MwDatetimeGet (
            uint32_t dtBin[2] )
```

Get date and time.

Parameters

out	*dtBin*	Date and time in seconds since Epoch. If set to NULL, this info is not filled (but return value will still be properly set).

Returns

A string with the date and time in textual format, e.g.: "Thu Mar 3 12:26:51 2016".

Definition at line 639 of file megawifi.c.

4.6.3.11 MwDefaultCfgSet()

```
int MwDefaultCfgSet (
            void  )
```

Set default module configuration.

Returns

MW_OK if configuration successfully reset, MW_ERROR otherwise.

Note

For this command to take effect, it must be followed by a module reset.

Definition at line 173 of file megawifi.c.

4.6.3.12 MwFlashRead()

```
uint8_t* MwFlashRead (
             uint32_t addr,
             uint16_t dataLen )
```

Read data from specified flash address.

Parameters

in	*addr*	Address from which data will be read.
in	*dataLen*	Number of bytes to read from addr.

Returns

Pointer to read data on success, or NULL if command failed.

Definition at line 731 of file megawifi.c.

4.6.3.13 MwFlashSectorErase()

```
int MwFlashSectorErase (
             uint16_t sect )
```

Erase a 4 KiB Flash sector. Every byte of an erased sector can be read as 0xFF.

Parameters

in	*sect*	Sector number to erase.

Returns

MW_OK if success, MW_ERROR if sector could not be erased.

Definition at line 687 of file megawifi.c.

4.6.3.14 MwFlashWrite()

```
int MwFlashWrite (
            uint32_t addr,
            uint8_t data[],
            uint16_t dataLen )
```

Write data to specified flash address.

Parameters

in	*addr*	Address to which data will be written.
in	*data*	Data to be written to flash chip.
in	*dataLen*	Length in bytes of data field.

Returns

MW_OK if success, MW_ERROR if data could not be written.

Definition at line 709 of file megawifi.c.

4.6.3.15 MwInit()

```
int MwInit (
            char * cmdBuf,
            uint16_t bufLen )
```

MwInit Module initialization. Must be called once before using any other function. It also initializes de UART.

Parameters

in	*cmdBuf*	Pointer to the buffer used to send and receive commands.
in	*bufLen*	Length of cmdBuf in bytes.

Returns

0 if Initialization successful, lower than 0 otherwise.

Definition at line 41 of file megawifi.c.

4.6.3.16 MwIpCfgGet()

```
int MwIpCfgGet (
            uint8_t index,
            MwIpCfg ** ip )
```

Get IPv4 configuration.

Parameters

in	*index*	Index of the configuration to get.
out	*ip*	Double pointer to MwIpCfg structure, with IP conf.

Returns

MW_OK if configuration successfully got, MW_ERROR otherwise.

Definition at line 282 of file megawifi.c.

4.6.3.17 MwIpCfgSet()

```
int MwIpCfgSet (
            uint8_t index,
            const MwIpCfg * ip )
```

Set IPv4 configuration.

Parameters

in	*index*	Index of the configuration to set.
in	*ip*	Pointer to the MwIpCfg structure, with IP configuration.

Returns

MW_OK if configuration successfully set, MW_ERROR otherwise.

Note

Strings must be NULL terminated. Maximum SSID length is 32 bytes, maximum pass length is 64 bytes.

Parameters

in	*index*	Index of the configuration to set.
in	*ip*	Pointer to the MwIpCfg structure, with IP configuration.

Returns

MW_OK if configuration successfully set, MW_ERROR otherwise.

Definition at line 256 of file megawifi.c.

4.6.3.18 MwRecv()

```
int MwRecv (
            uint8_t ** data,
            uint16_t * len,
            uint32_t maxLoopCnt )
```

Receive data.

Parameters

out	*data*	Double pointer to received data.
out	*len*	Length of the received data.
in	*maxLoopCnt*	Maximum number of iterations to try before giving up. Set to 0 to avoid waiting if no data available.

Returns

On success, channel on which data has been received, or MW_ERROR if no data was received.

Definition at line 517 of file megawifi.c.

4.6.3.19 MwSntpCfgSet()

```
int MwSntpCfgSet (
            char * servers[3],
            uint8_t upDelay,
            char timezone,
            char dst )
```

Configure SNTP parameters and timezone.

Parameters

in	*servers*	Array of up to three NTP servers. If less than three servers are desired, unused entries must be empty.
in	*upDelay*	Update delay in seconds. Minimum value is 15.
in	*timezone*	Time zone information (from -11 to 13).
in	*dst*	Daylight saving. Set to 1 to apply 1 hour offset.

Returns

MW_OK on success, MW_ERROR if command fails.

Definition at line 604 of file megawifi.c.

4.6.3.20 MwSockStatGet()

```
MwSockStat MwSockStatGet (
            uint8_t ch )
```

Get socket status.

Parameters

in	*ch*	Channel associated to the socket asked for status.

Returns

Socket status data on success, or MW_ERROR on error.

Definition at line 579 of file megawifi.c.

4.6.3.21 MwSysStatGet()

```
MwMsgSysStat* MwSysStatGet (
            void  )
```

Get system status.

Returns

Pointer to system status structure on success, or NULL on error.

Definition at line 561 of file megawifi.c.

4.6.3.22 MwTcpBind()

```
int MwTcpBind (
            uint8_t ch,
            uint16_t port )
```

Binds a socket to a port, and listens to connections on the port. If a connection request is received, it will be automatically accepted.

Parameters

in	*ch*	Channel associated to the socket bound t port.
in	*port*	Port number to which the socket will be bound.

Returns

MW_OK if socket successfully bound, or MW_ERROR if command failed.

Definition at line 456 of file megawifi.c.

4.6.3.23 MwTcpConnect()

```
int MwTcpConnect (
            uint8_t ch,
            char dstaddr[],
            char dstport[],
            char srcport[] )
```

Tries establishing a TCP connection with specified server.

Parameters

in	*ch*	Channel used for the connection.
in	*dstaddr*	Address (IP or DNS entry) of the server.
in	*dstport*	Port in which server is listening.
in	*srcport*	Port from which try establishing connection. Set to 0 or empty string for automatic port allocation.

Returns

MW_OK if connection successfully established, or MW_ERROR if connection failed.

Definition at line 396 of file megawifi.c.

4.6.3.24 MwTcpDisconnect()

```
int MwTcpDisconnect (
            uint8_t ch )
```

Disconnects a TCP socket from specified channel.

Parameters

in	*ch*	Channel associated to the socket to disconnect.

Returns

MW_OK if socket successfully disconnected, or MW_ERROR if command failed.

Definition at line 430 of file megawifi.c.

4.6.3.25 MwVersionGet()

```
int MwVersionGet (
            uint8_t * verMajor,
            uint8_t * verMinor,
            char * variant[] )
```

Obtain module version numbers and string.

Parameters

out	*verMajor*	Pointer to Major version number.
out	*verMinor*	Pointer to Minor version number.
out	*variant*	String with firmware variant ("std" for standard).

Returns

MW_OK if completed successfully, MW_ERROR otherwise.

Definition at line 118 of file megawifi.c.

4.7 mwmsg

MeGaWiFi command message definitions. Contains the definition of the command codes and the data structures conforming the command message queries and responses.

Modules

- MwCmds

  Supported commands.

Data Structures

- struct MwMsgInAddr

  TCP/UDP address message.
- struct MwIpCfg

  IP configuration parameters.
- struct MwMsgApCfg

  AP configuration message.
- struct MwMsgIpCfg

  IP configuration message.
- struct MwMsgSntpCfg

  SNTP and timezone configuration.
- struct MwMsgDateTime

  Date and time message.
- struct MwMsgFlashData

  Flash memory address and data.
- struct MwMsgFlashRange

  Flash memory block.
- struct MwMsgBind

  Bind message data.
- union MwMsgSysStat

  System status.
- struct MwCmd

  Command sent to system FSM.

Macros

- #define MW_MSG_MAX_BUFLEN 512

  Maximum buffer length (bytes)
- #define MW_CMD_HEADLEN (2 ∗ sizeof(uint16_t))

  Command header length (command code and data length fields).
- #define MW_CMD_MAX_BUFLEN (MW_MSG_MAX_BUFLEN - MW_CMD_HEADLEN)

  Maximum data length contained inside command buffer.
- #define MW_SSID_MAXLEN 32

  Maximum SSID length (including '\0').
- #define MW_PASS_MAXLEN 64

  Maximum password length (including '\0').

Enumerations

- enum MwState {
  MW_ST_INIT = 0, MW_ST_IDLE, MW_ST_AP_JOIN, MW_ST_SCAN,
  MW_ST_READY, MW_ST_TRANSPARENT, MW_ST_MAX }

  MwState Possible states of the system state machine.
- enum MwSockStat { MW_SOCK_NONE = 0, MW_SOCK_TCP_LISTEN, MW_SOCK_TCP_EST, MW_SOCK_UDP_READY }

  Socket status.

4.7.1 Detailed Description

MeGaWiFi command message definitions. Contains the definition of the command codes and the data structures conforming the command message queries and responses.

Author

Jesus Alonso (doragasu)

Date

2015

4.7.2 Enumeration Type Documentation

4.7.2.1 MwSockStat

`enum MwSockStat`

Socket status.

Enumerator

MW_SOCK_NONE	Unused socket.
MW_SOCK_TCP_LISTEN	Socket bound and listening.
MW_SOCK_TCP_EST	TCP socket, connection established.
MW_SOCK_UDP_READY	UDP socket ready for sending/receiving.

Definition at line 149 of file mw-msg.h.

4.7.2.2 MwState

`enum MwState`

MwState Possible states of the system state machine.

Enumerator

MW_ST_INIT	Initialization state.
MW_ST_IDLE	Idle state, until connected to an AP.
MW_ST_AP_JOIN	Trying to join an access point.
MW_ST_SCAN	Scanning access points.
MW_ST_READY	Connected to The Internet.
MW_ST_TRANSPARENT	Transparent communication state.
MW_ST_MAX	Limit number for state machine.

Definition at line 138 of file mw-msg.h.

4.8 sysfsm

System controller for wflash. Keeps the system status and processes incoming events, to perform the requested actions.

Macros

- #define SF_ENTRY_POINT_ADDR (*((uint32_t*)0x1C8))

Functions

- void SfInit (void)
- int SfCycle (void)
- void SfBoot (uint32_t addr)

4.8.1 Detailed Description

System controller for wflash. Keeps the system status and processes incoming events, to perform the requested actions.

Author

Jesús Alonso (doragasu)

Date

2017

Todo Module currently does not support checksum/CRC

4.8.2 Macro Definition Documentation

4.8.2.1 SF_ENTRY_POINT_ADDR

```
#define SF_ENTRY_POINT_ADDR (*((uint32_t*)0x1C8))
```

Entry point address is stored at the beginning of the NOTES section of the cartridge header

Definition at line 19 of file sysfsm.h.

4.8.3 Function Documentation

4.8.3.1 SfBoot()

```
void SfBoot (
            uint32_t addr )
```

Clear environment and boot from specified address.

Parameters

in	*addr*	Address from which to boot.

Definition at line 298 of file sysfsm.c.

4.8.3.2 SfCycle()

```
int SfCycle (
            void  )
```

Perform one cycle of the system state machine.

Returns

0 if OK, non-zero if error.

Definition at line 83 of file sysfsm.c.

4.8.3.3 SfInit()

```
void SfInit (
            void  )
```

Module initialization. Call this function before using this module.

Definition at line 36 of file sysfsm.c.

4.9 vdp

Basic VDP handling routines. This module implements basic VDP related routines for:

Modules

- VdpIoPortAddr

  Addresses for the VDP IO ports.
- VdpIoPort

  VDP control and data ports.
- VdpColors

  Simple color definitions in CRAM format.
- VdpNametableAddr

  Nametable addresses in VRAM.
- VdpTextColors

  Available text colors, to use with VdpDrawText() and VdpDrawHex() calls.

Macros

- #define VdpColor(r, g, b) (((r)<<1) | ((g)<<5) | ((b)<<9))

  Build color in CRAM format, with 3-bit r, b and b components.
- #define VDP_STAT_VBLANK 0x0008

  Flag of the status register corresponging to the VBLANK interval.

Enumerations

- enum {
  VDP_VRAM_RD = 0, VDP_VRAM_WR, VDP_CRAM_RD, VDP_CRAM_WR,
  VDP_VSRAM_RD, VDP_VSRAM_WR, VDP_VRAM_RD_8B, VDP_RAM_TYPE_MAX }

  RAM types managed by the VDP.
- enum VdpReg {
  VDP_REG_MODE1 = 0, VDP_REG_MODE2, VDP_REG_PLANEA_NT, VDP_REG_WIN_NT,
  VDP_REG_PLANEB_NT, VDP_REG_SPR_T, VDP_REG_SPR_PGADDR, VDP_REG_BGCOL,
  VDP_REG_UNUSED1, VDP_REG_UNUSED2, VDP_REG_HINT_CNT, VDP_REG_MODE3,
  VDP_REG_MODE4, VDP_REG_HSCROLL, VDP_REG_NT_ADDR, VDP_REG_INCR,
  VDP_REG_PSIZE, VDP_REG_WIN_HPOS, VDP_REG_WIN_VPOS, VDP_REG_DMALEN1,
  VDP_REG_DMALEN2, VDP_REG_DMASRC1, VDP_REG_DMASRC2, VDP_REG_DMASRC3,
  VDP_REG_MAX }

  VDP registers.

Functions

- void VdpInit (void)
- void VdpFontLoad (const uint32_t font[ ], uint8_t chars, uint16_t addr, uint8_t fgcol, uint8_t bgcol)
- void VdpVRamClear (uint16_t addr, uint16_t wlen)
- void VdpLineClear (uint16_t planeAddr, uint8_t line)
- void VdpDrawText (uint16_t planeAddr, uint8_t x, uint8_t y, uint8_t txtColor, char text[ ])
- void VdpDrawHex (uint16_t planeAddr, uint8_t x, uint8_t y, uint8_t txtColor, uint8_t num)
- uint8_t VdpDrawDec (uint16_t planeAddr, uint8_t x, uint8_t y, uint8_t txtColor, uint8_t num)
- void VdpVBlankWait (void)

Variables

- const uint16_t cdMask [VDP_RAM_TYPE_MAX]

  Mask for reading/writing from/to VDP memory.

4.9.1 Detailed Description

Basic VDP handling routines. This module implements basic VDP related routines for:

- VDP initialization.
- Font loading and colour text drawing on planes. No sprites or any other fancy stuff.

Author

Jesús Alonso (doragasu)

Date

2017

4.9.2 Enumeration Type Documentation

4.9.2.1 anonymous enum

`anonymous enum`

RAM types managed by the VDP.

Enumerator

VDP_VRAM_RD	VRAM read.
VDP_VRAM_WR	VRAM write.
VDP_CRAM_RD	CRAM read.
VDP_CRAM_WR	CRAM write.
VDP_VSRAM_RD	VSRAM read.
VDP_VSRAM_WR	VSRAM write.
VDP_VRAM_RD_8B	VRAM 8-bit read (undocumented)
VDP_RAM_TYPE_MAX	Limit (do not use)

Definition at line 92 of file vdp.h.

4.9.2.2 VdpReg

```
enum VdpReg
```

VDP registers.

Enumerator

VDP_REG_MODE1	Mode set register #1.
VDP_REG_MODE2	Mode set register #2.
VDP_REG_PLANEA_NT	Plane A pattern name table.
VDP_REG_WIN_NT	Window pattern name table.
VDP_REG_PLANEB_NT	Plane A pattern name table.
VDP_REG_SPR_T	Sprite attribute table base address.
VDP_REG_SPR_PGADDR	Sprite pattern generator base address.
VDP_REG_BGCOL	Background colour.
VDP_REG_UNUSED1	Unused.
VDP_REG_UNUSED2	Unused.
VDP_REG_HINT_CNT	H-Interrupt register.
VDP_REG_MODE3	Mode set register #3.
VDP_REG_MODE4	Mode set register #4.
VDP_REG_HSCROLL	H scroll data table base address.
VDP_REG_NT_ADDR	Nametable pattern generator base address.
VDP_REG_INCR	Auto increment data.
VDP_REG_PSIZE	Plane size.
VDP_REG_WIN_HPOS	Window H position.
VDP_REG_WIN_VPOS	Window V position.
VDP_REG_DMALEN1	DMA Length register #1.
VDP_REG_DMALEN2	DMA Length register #2.
VDP_REG_DMASRC1	DMA source register #1.
VDP_REG_DMASRC2	DMA source register #2.
VDP_REG_DMASRC3	DMA source register #3.
VDP_REG_MAX	Limit (do not use)

Definition at line 107 of file vdp.h.

4.9.3 Function Documentation

4.9.3.1 VdpDrawDec()

```
uint8_t VdpDrawDec (
            uint16_t planeAddr,
            uint8_t x,
            uint8_t y,
            uint8_t txtColor,
            uint8_t num )
```

Draws an 8-bit decimal number on a plane.

Parameters

in	*planeAddr*	Address in VRAM of the plane used to draw text.
in	*x*	Horizontal text coordinate.
in	*y*	Vertical text coordinate.
in	*txtColor*	Text colour (see VdpTextColors).
in	*num*	Number to draw on the plane in hexadecimal format.

Returns

Number of characters used by the drawn number.

Definition at line 188 of file vdp.c.

4.9.3.2 VdpDrawHex()

```
void VdpDrawHex (
            uint16_t planeAddr,
            uint8_t x,
            uint8_t y,
            uint8_t txtColor,
            uint8_t num )
```

Draws an 8-bit hexadecimal number on a plane.

Parameters

in	*planeAddr*	Address in VRAM of the plane used to draw text.
in	*x*	Horizontal text coordinate.
in	*y*	Vertical text coordinate.
in	*txtColor*	Text colour (see VdpTextColors).
in	*num*	Number to draw on the plane in hexadecimal format.

Definition at line 147 of file vdp.c.

4.9.3.3 VdpDrawText()

```
void VdpDrawText (
            uint16_t planeAddr,
            uint8_t x,
            uint8_t y,
            uint8_t txtColor,
            char text[] )
```

Draws text on a plane.

Parameters

in	*planeAddr*	Address in VRAM of the plane used to draw text.
in	*x*	Horizontal text coordinate.
in	*y*	Vertical text coordinate.
in	*txtColor*	Text colour (see VdpTextColors).
in	*text*	Null terminated text to write to the plane.

Definition at line 124 of file vdp.c.

4.9.3.4 VdpFontLoad()

```
void VdpFontLoad (
            const uint32_t font[],
            uint8_t chars,
            uint16_t addr,
            uint8_t fgcol,
            uint8_t bgcol )
```

Loads a 1bpp font on the VRAM, setting specified foreground and background colours.

Parameters

in	*font*	Array containing the 1bpp font (8 bytes per character).
in	*chars*	Number of characters contained in font.
in	*addr*	VRAM Address to load the font in.
in	*fgcol*	Foreground colour, in CRAM colour format.
in	*bgcol*	Background colour, in CRAM colour format.

Definition at line 232 of file vdp.c.

4.9.3.5 VdpInit()

```
void VdpInit (
            void  )
```

VDP Initialization. Call this function once before using this module.

Definition at line 47 of file vdp.c.

4.9.3.6 VdpLineClear()

```
void VdpLineClear (
            uint16_t planeAddr,
            uint8_t line )
```

Clears (sets to 0) the plane line.

Parameters

in	*planeAddr*	Address in VRAM of the plane to clear.
in	*line*	Line number to clear.

Definition at line 261 of file vdp.c.

4.9.3.7 VdpVBlankWait()

```
void VdpVBlankWait (
            void  )
```

Waits until the beginning of the next VBLANK cycle.

Definition at line 273 of file vdp.c.

4.9.3.8 VdpVRamClear()

```
void VdpVRamClear (
            uint16_t addr,
            uint16_t wlen )
```

Clears (sets to 0) the specified VRAM range.

Parameters

in	*addr*	VRAM address to clear.
in	*wlen*	Length in words of the range to clear.

Definition at line 169 of file vdp.c.

4.9.4 Variable Documentation

4.9.4.1 cdMask

```
const uint16_t cdMask[VDP_RAM_TYPE_MAX]
```

Mask for reading/writing from/to VDP memory.

Mask used to build control port data for VDP RAM writes. Bits 15 and 14: CD1 and CD0. Bits 7 to 4: CD5 to CD2.

Definition at line 20 of file vdp.c.

4.10 WFlash

WFlash command definitions.

Data Structures

- struct WfMemRange

  Memory range definition.
- struct WfCmd

  Command definition.
- union WfBuf

  Command buffer. Allows accessing the buffer as a command or as raw data.

Macros

- #define WF_VERSION_MAJOR 0x00

  Major number of the commands implementation.
- #define WF_VERSION_MINOR 0x01

  Minor number of the commands implementation.
- #define WF_MAX_DATALEN 1440

  Maximum payload length.
- #define WF_HEADLEN 4

  Header lenght.
- #define WF_CHANNEL 1

  MegaWiFi channel.
- #define WF_CMD_OK 0

  OK reply code to a command.
- #define WF_CMD_ERROR 1

  ERROR reply code to a command.

Enumerations

- enum {
  WF_CMD_VERSION_GET = 0, WF_CMD_ECHO, WF_CMD_ID_GET, WF_CMD_ERASE,
  WF_CMD_PROGRAM, WF_CMD_READ, WF_CMD_RUN, WF_CMD_AUTORUN,
  WF_CMD_MAX }

  Available commands.

4.10.1 Detailed Description

WFlash command definitions.

Author

Jesús Alonso (doragasu)

Date

2017

4.10.2 Enumeration Type Documentation

4.10.2.1 anonymous enum

```
anonymous enum
```

Available commands.

Enumerator

WF_CMD_VERSION_GET	Get bootloader version.
WF_CMD_ECHO	Echo data.
WF_CMD_ID_GET	Get flash chip ID.
WF_CMD_ERASE	Erase range.
WF_CMD_PROGRAM	Program data.
WF_CMD_READ	Read data.
WF_CMD_RUN	Run from address.
WF_CMD_AUTORUN	Run from entry point in cart header.
WF_CMD_MAX	Maximum command value delimiter.

Definition at line 27 of file cmds.h.

4.11 GpRegAddrs

Gamepad related register addresses.

Macros

- #define GP_REG_VERSION_ADDR 0xA10001
  Version number address.
- #define GP_REG_PORTA_DATA_ADDR 0xA10003
  Port A, data register address.
- #define GP_REG_PORTB_DATA_ADDR 0xA10005
  Port B, data register address.
- #define GP_REG_PORTC_DATA_ADDR 0xA10007
  Port C, data register address.
- #define GP_REG_PORTA_CTRL_ADDR 0xA10009
  Port A, control register address.
- #define GP_REG_PORTB_CTRL_ADDR 0xA1000B
  Port B, control register address.
- #define GP_REG_PORTC_CTRL_ADDR 0xA1000D
  Port C, control register address.

4.11.1 Detailed Description

Gamepad related register addresses.

4.12 GpRegs

Gamepad related registers.

Macros

- #define GP_REG_VERSION (∗((volatile uint8_t∗)GP_REG_VERSION_ADDR))
  Gamepad hardware version number.
- #define GP_REG_PORTA_DATA (∗((volatile uint8_t∗)GP_REG_PORTA_DATA_ADDR))
  Port A, data register.
- #define GP_REG_PORTB_DATA (∗((volatile uint8_t∗)GP_REG_PORTB_DATA_ADDR))
  Port B, data register.
- #define GP_REG_PORTC_DATA (∗((volatile uint8_t∗)GP_REG_PORTC_DATA_ADDR))
  Port C, data register.
- #define GP_REG_PORTA_CTRL (∗((volatile uint8_t∗)GP_REG_PORTA_CTRL_ADDR))
  Port A, control register.
- #define GP_REG_PORTB_CTRL (∗((volatile uint8_t∗)GP_REG_PORTB_CTRL_ADDR))
  Port B, control register.
- #define GP_REG_PORTC_CTRL (∗((volatile uint8_t∗)GP_REG_PORTC_CTRL_ADDR))
  Port C, control register.

4.12.1 Detailed Description

Gamepad related registers.

4.13 GpMasks

Masks used to filter the cross and buttons.

Macros

- #define GP_START_MASK 0x80

  Start button.
- #define GP_A_MASK 0x40

  A button.
- #define GP_B_MASK 0x10

  B button.
- #define GP_C_MASK 0x20

  C button.
- #define GP_RIGHT_MASK 0x08

  Right direction.
- #define GP_LEFT_MASK 0x04

  Left direction.
- #define GP_DOWN_MASK 0x02

  Down direction.
- #define GP_UP_MASK 0x01

  Up direction.

4.13.1 Detailed Description

Masks used to filter the cross and buttons.

4.14 UartRegs

16C550 UART registers

Macros

- #define UART_RHR (∗((volatile uint8_t∗)(UART_BASE + 0)))
  Receiver holding register. Read only.
- #define UART_THR (∗((volatile uint8_t∗)(UART_BASE + 0)))
  Transmit holding register. Write only.
- #define UART_IER (∗((volatile uint8_t∗)(UART_BASE + 2)))
  Interrupt enable register. Write only.
- #define UART_FCR (∗((volatile uint8_t∗)(UART_BASE + 4)))
  FIFO control register. Write only.
- #define UART_ISR (∗((volatile uint8_t∗)(UART_BASE + 4)))
  Interrupt status register. Read only.
- #define UART_LCR (∗((volatile uint8_t∗)(UART_BASE + 6)))
  Line control register. Write only.
- #define UART_MCR (∗((volatile uint8_t∗)(UART_BASE + 8)))
  Modem control register. Write only.
- #define UART_LSR (∗((volatile uint8_t∗)(UART_BASE + 10)))
  Line status register. Read only.
- #define UART_MSR (∗((volatile uint8_t∗)(UART_BASE + 12)))
  Modem status register. Read only.
- #define UART_SPR (∗((volatile uint8_t∗)(UART_BASE + 14)))
  Scratchpad register.
- #define UART_DLL (∗((volatile uint8_t∗)(UART_BASE + 0)))
  Divisor latch LSB. Acessed only when LCR[7] = 1.
- #define UART_DLM (∗((volatile uint8_t∗)(UART_BASE + 2)))
  Divisor latch MSB. Acessed only when LCR[7] = 1.

4.14.1 Detailed Description

16C550 UART registers

Note

Do NOT access IER, FCR, LCR and MCR directly, use Set/Get functions. Remaining registers can be directly accessed, but meeting the read only/write only restrictions.

4.15 UartOuts

Output pins controlled by the MCR UART register.

Macros

- #define UART_MCR__DTR 0x01

  Data Terminal Ready.
- #define UART_MCR__RTS 0x02

  Request To Send.
- #define UART_MCR__OUT1 0x04

  GPIO pin 1.
- #define UART_MCR__OUT2 0x08

  GPIO pin 2.

4.15.1 Detailed Description

Output pins controlled by the MCR UART register.

4.16 UartIns

Input pins readed in the MSR UART register.

Macros

- #define UART_MSR__DSR 0x20
  Data Set Ready.

4.16.1 Detailed Description

Input pins readed in the MSR UART register.

4.17 ReturnCodes

OK/Error codes returned by several functions.

Macros

- #define LSD_OK 0

  Function completed successfully.
- #define LSD_ERROR -1

  Generic error code.
- #define LSD_FRAMING_ERROR -2

  A framing error occurred. Possible data loss.

4.17.1 Detailed Description

OK/Error codes returned by several functions.

4.18 MwCtrlPins

Pins used to control WiFi module.

Macros

- #define MW__RESET UART_MCR__OUT1

  Reset out.
- #define MW__PRG UART_MCR__OUT2

  Program out.
- #define MW__PD UART_MCR__DTR

  Power Down out.
- #define MW__DAT UART_MSR__DSR

  Data request in.

4.18.1 Detailed Description

Pins used to control WiFi module.

4.19 MwRetVals

Function return values.

Macros

- #define MW_OK 0

  The function completed successfully.
- #define MW_ERROR -1

  The function completed with error.

4.19.1 Detailed Description

Function return values.

4.20 MwCmds

Supported commands.

Macros

- #define MW_CMD_OK 0
  OK command reply.
- #define MW_CMD_VERSION 1
  Get firmware version.
- #define MW_CMD_ECHO 2
  Echo data.
- #define MW_CMD_AP_SCAN 3
  Scan for access points.
- #define MW_CMD_AP_CFG 4
  Configure access point.
- #define MW_CMD_AP_CFG_GET 5
  Get access point configuration.
- #define MW_CMD_IP_CFG 6
  Configure IPv4.
- #define MW_CMD_IP_CFG_GET 7
  Get IPv4 configuration.
- #define MW_CMD_AP_JOIN 8
  Join access point.
- #define MW_CMD_AP_LEAVE 9
  Leave previously joined access point.
- #define MW_CMD_TCP_CON 10
  Connect TCP socket.
- #define MW_CMD_TCP_BIND 11
  Bind TCP socket to port.
- #define MW_CMD_TCP_ACCEPT 12
  Accept incomint TCP connection.
- #define MW_CMD_TCP_DISC 13
  Disconnect and free TCP socket.
- #define MW_CMD_UDP_SET 14
  Configure UDP socket.
- #define MW_CMD_UDP_CLR 15
  Clear and free UDP socket.
- #define MW_CMD_SOCK_STAT 16
  Get socket status.
- #define MW_CMD_PING 17
  Ping host.
- #define MW_CMD_SNTP_CFG 18
  Configure SNTP service.
- #define MW_CMD_DATETIME 19
  Get date and time.
- #define MW_CMD_DT_SET 20
  Set date and time.
- #define MW_CMD_FLASH_WRITE 21

Write to WiFi module flash.

- #define MW_CMD_FLASH_READ 22

  Read from WiFi module flash.

- #define MW_CMD_FLASH_ERASE 23

  Erase sector from WiFi flash.

- #define MW_CMD_FLASH_ID 24

  Get WiFi flash chip identifiers.

- #define MW_CMD_SYS_STAT 25

  Get system status.

- #define MW_CMD_DEF_CFG_SET 26

  Set default configuration.

- #define MW_CMD_HRNG_GET 27

  Gets random numbers.

- #define MW_CMD_ERROR 255

  Error command reply.

4.20.1 Detailed Description

Supported commands.

4.21 VdploPortAddr

Addresses for the VDP IO ports.

Macros

- #define VDP_DATA_PORT_ADDR 0xC00000

  VDP Data port address.
- #define VDP_CTRL_PORT_ADDR 0xC00004

  VDP Control port address.
- #define VDP_HV_COUNT_ADDR 0xC00008

  VDP scanline counter address.

4.21.1 Detailed Description

Addresses for the VDP IO ports.

4.22 VdploPort

VDP control and data ports.

Macros

- #define VDP_DATA_PORT_W (∗((volatile uint16_t∗)VDP_DATA_PORT_ADDR))
  VDP data port, WORD access.
- #define VDP_DATA_PORT_DW (∗((volatile uint32_t∗)VDP_DATA_PORT_ADDR))
  VDP data port, DWORD access.
- #define VDP_CTRL_PORT_W (∗((volatile uint16_t∗)VDP_CTRL_PORT_ADDR))
  VDP control port, WORD access.
- #define VDP_CTRL_PORT_DW (∗((volatile uint32_t∗)VDP_CTRL_PORT_ADDR))
  VDP control port, DWORD access.
- #define VDP_HV_COUNT_W (∗((volatile uint16_t∗)VDP_HV_COUNT_ADDR))
  VDP scanline counter port, WORD access.

4.22.1 Detailed Description

VDP control and data ports.

4.23 VdpColors

Simple color definitions in CRAM format.

Macros

- #define VDP_COLOR_BLACK VdpColor(0, 0, 0)
  Black color.
- #define VDP_COLOR_RED VdpColor(7, 0, 0)
  Red color.
- #define VDP_COLOR_GREEN VdpColor(0, 7, 0)
  Green color.
- #define VDP_COLOR_BLUE VdpColor(0, 0, 7)
  Blue color.
- #define VDP_COLOR_CYAN VdpColor(0, 7, 7)
  Cyan color.
- #define VDP_COLOR_MAGENTA VdpColor(7, 0, 7)
  Magenta color.
- #define VDP_COLOR_YELLOW VdpColor(7, 7, 0)
  Yellow color.
- #define VDP_COLOR_WHITE VdpColor(7, 7, 7)
  White color.

4.23.1 Detailed Description

Simple color definitions in CRAM format.

4.24 VdpNametableAddr

Nametable addresses in VRAM.

Macros

- #define VDP_PLANEA_ADDR 0x4000

  PLANE A nametable address.
- #define VDP_PLANEB_ADDR 0x6000

  PLANE B nametable address.
- #define VDP_WIN_ADDR 0x8000

  WINDOW nametable address.

4.24.1 Detailed Description

Nametable addresses in VRAM.

4.25 VdpTextColors

Available text colors, to use with VdpDrawText() and VdpDrawHex() calls.

Macros

- #define VDP_TXT_COL_WHITE 0x00

  White text color.
- #define VDP_TXT_COL_CYAN 0x60

  Cyan text color.
- #define VDP_TXT_COL_MAGENTA 0xC0

  Magenta text color.

4.25.1 Detailed Description

Available text colors, to use with VdpDrawText() and VdpDrawHex() calls.

Chapter 5

Data Structure Documentation

5.1 FlashCmd Struct Reference

Data used to perform different flash commands.

```
#include <flash.h>
```

Data Fields

- uint16_t addr

  Flash address.
- uint8_t data

  Flash data.

5.1.1 Detailed Description

Data used to perform different flash commands.

Definition at line 23 of file flash.h.

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

- flash.h

5.2 LsdData Struct Reference

Local data required by the module.

Data Fields

- LsdState rxs
  Reception state.
- LsdState txs
  Send state.
- uint8_t en [LSD_MAX_CH]
  Channel enable.
- uint16_t pos
  Position in current buffer.
- uint8_t current
  Current buffer in use.

5.2.1 Detailed Description

Local data required by the module.

Definition at line 34 of file lsd.c.

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

- mw/lsd.c

5.3 MwApData Struct Reference

Access Point data.

```
#include <megawifi.h>
```

Data Fields

- char auth
  Authentication type.
- char channel
  WiFi channel.
- char str
  Signal strength.
- char ssidLen
  Length of ssid field.
- char * ssid
  SSID string (not NULL terminated).

5.3.1 Detailed Description

Access Point data.

Definition at line 67 of file megawifi.h.

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

- mw/megawifi.h

5.4 MwCmd Struct Reference

Command sent to system FSM.

```
#include <mw-msg.h>
```

Data Fields

- uint16_t cmd
  Command code.
- uint16_t datalen
  Data length.
- union {
  uint8_t ch
  uint8_t data [MW_CMD_MAX_BUFLEN]
  RAW data in uint8_t format.
  uint32_t dwData [MW_CMD_MAX_BUFLEN/sizeof(uint32_t)]
  RAW data in uint32_t format.
  MwMsgInAddr inAddr
  Internet address.
  MwMsgApCfg apCfg
  Access Point configuration.
  MwMsgIpCfg ipCfg
  IP configuration.
  MwMsgSntpCfg sntpCfg
  SNTP client configuration.
  MwMsgDateTime datetime
  Date and time message.
  MwMsgFlashData flData
  Flash memory data.
  MwMsgFlashRange flRange
  Flash memory range.
  MwMsgBind bind
  Bind message.
  MwMsgSysStat sysStat
  System status.
  uint16_t flSect
  Flash sector.
  uint32_t flId
  Flash IDs.
  uint16_t rndLen
  Length of the random buffer to fill.
  };

5.4.1 Detailed Description

Command sent to system FSM.

Definition at line 171 of file mw-msg.h.

5.4.2 Field Documentation

5.4.2.1 ch

```
uint8_t MwCmd::ch
```

Channel number for channel related requests

Definition at line 177 of file mw-msg.h.

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

- mw/mw-msg.h

5.5 MwIpCfg Struct Reference

IP configuration parameters.

```
#include <mw-msg.h>
```

Data Fields

- uint32_t addr
  Host IP address in binary format.
- uint32_t mask
  Subnet mask in binary IP format.
- uint32_t gateway
  Gateway IP address in binary format.
- uint32_t dns1
  DNS server 1 IP address in binary format.
- uint32_t dns2
  DNS server 2 IP address in binary format.

5.5.1 Detailed Description

IP configuration parameters.

Definition at line 75 of file mw-msg.h.

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

- mw/mw-msg.h

5.6 MwMsgApCfg Struct Reference

AP configuration message.

`#include <mw-msg.h>`

Data Fields

- uint8_t cfgNum
  Configuration number.
- char ssid [MW_SSID_MAXLEN]
  SSID string.
- char pass [MW_PASS_MAXLEN]
  Password string.

5.6.1 Detailed Description

AP configuration message.

Warning

If ssid length is MW_SSID_MAXLEN, the string will not be NULL terminated. Also if pass length equals MW_PASS_MAXLEN, pass

Definition at line 87 of file mw-msg.h.

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

- mw/mw-msg.h

5.7 MwMsgBind Struct Reference

Bind message data.

`#include <mw-msg.h>`

Data Fields

- uint32_t reserved
  Reserved, set to 0.
- uint16_t port
  Port to bind to.
- uint8_t channel
  Channel used for the socket bound to port.

5.7.1 Detailed Description

Bind message data.

Definition at line 131 of file mw-msg.h.

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

- mw/mw-msg.h

5.8 MwMsgDateTime Struct Reference

Date and time message.

```
#include <mw-msg.h>
```

Data Fields

- uint32_t dtBin [2]
- char dtStr [MW_CMD_MAX_BUFLEN - sizeof(uint64_t)]

  Date and time in textual format.

5.8.1 Detailed Description

Date and time message.

Definition at line 111 of file mw-msg.h.

5.8.2 Field Documentation

5.8.2.1 dtBin

```
uint32_t MwMsgDateTime::dtBin[2]
```

Number of seconds since Epoch (64-bit)

Definition at line 112 of file mw-msg.h.

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

- mw/mw-msg.h

5.9 MwMsgFlashData Struct Reference

Flash memory address and data.

`#include <mw-msg.h>`

Data Fields

- uint32_t addr
- uint8_t data [MW_CMD_MAX_BUFLEN - sizeof(uint32_t)]
  Data associated to the address.

5.9.1 Detailed Description

Flash memory address and data.

Definition at line 118 of file mw-msg.h.

5.9.2 Field Documentation

5.9.2.1 addr

`uint32_t MwMsgFlashData::addr`

Flash memory address

Definition at line 119 of file mw-msg.h.

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

- mw/mw-msg.h

5.10 MwMsgFlashRange Struct Reference

Flash memory block.

`#include <mw-msg.h>`

Data Fields

- uint32_t addr
  Start address.
- uint16_t len
  Length of the block.

5.10.1 Detailed Description

Flash memory block.

Definition at line 125 of file mw-msg.h.

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

- mw/mw-msg.h

5.11 MwMsgInAddr Struct Reference

TCP/UDP address message.

`#include <mw-msg.h>`

Data Fields

- char dst_port [6]
  TCP destination port string.
- char src_port [6]
  TCP source port string.
- uint8_t channel
- char dstAddr [MW_CMD_MAX_BUFLEN - 6 - 6 - 1]
  Data payload.

5.11.1 Detailed Description

TCP/UDP address message.

Definition at line 66 of file mw-msg.h.

5.11.2 Field Documentation

5.11.2.1 channel

`uint8_t MwMsgInAddr::channel`

LSD channel used for communications

Definition at line 69 of file mw-msg.h.

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

- mw/mw-msg.h

5.12 MwMsgIpCfg Struct Reference

IP configuration message.

`#include <mw-msg.h>`

Data Fields

- uint8_t cfgNum
  Configuration number.
- uint8_t reserved [3]
  Reserved (set to 0)
- MwIpCfg ip
  IPv4 configuration data.

5.12.1 Detailed Description

IP configuration message.

Definition at line 94 of file mw-msg.h.

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

- mw/mw-msg.h

5.13 MwMsgSntpCfg Struct Reference

SNTP and timezone configuration.

`#include <mw-msg.h>`

Data Fields

- uint16_t upDelay
  Update delay in seconds (min: 15)
- int8_t tz
  Timezone (from -11 to 13)
- uint8_t dst
- char servers [MW_CMD_MAX_BUFLEN - 4]

5.13.1 Detailed Description

SNTP and timezone configuration.

Definition at line 101 of file mw-msg.h.

5.13.2 Field Documentation

5.13.2.1 dst

```
uint8_t MwMsgSntpCfg::dst
```

Daylight savines (set to 1 to add 1 hour)

Definition at line 104 of file mw-msg.h.

5.13.2.2 servers

```
char MwMsgSntpCfg::servers[MW_CMD_MAX_BUFLEN - 4]
```

Up to 3 NTP server URLs, separated by a NULL character. A double NULL marks the end of the server list.

Definition at line 107 of file mw-msg.h.

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

- mw/mw-msg.h

5.14 MwMsgSysStat Union Reference

System status.

```
#include <mw-msg.h>
```

Data Fields

- uint32_t st_flags
  Accesses all the flags at once.
- struct {
  MwState sys_stat:8
  System status.
  uint8_t online:1
  Module is connected to the Internet.
  uint8_t cfg_ok:1
  Configuration OK.
  uint8_t dt_ok:1
  Date and time synchronized at least once.
  uint8_t cfg:2
  Network configuration set.
  uint16_t reserved:3
  Reserved flags.
  uint16_t ch_ev:16
  Channel flags with the pending event.
  };

5.14.1 Detailed Description

System status.

Definition at line 157 of file mw-msg.h.

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

- mw/mw-msg.h

5.15 SfData Struct Reference

Local module data structure.

Data Fields

- uint32_t waddr
  Word address to which write.
- uint32_t wrem
  Remaining words to write.
- SfStat s
  System status.

5.15.1 Detailed Description

Local module data structure.

Definition at line 24 of file sysfsm.c.

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

- sysfsm.c

5.16 UartShadow Struct Reference

Structure with the shadow registers.

```
#include <16c550.h>
```

Data Fields

- uint8_t IER
  Interrupt Enable Register.
- uint8_t FCR
  FIFO Control Register.
- uint8_t LCR
  Line Control Register.
- uint8_t MCR
  Modem Control Register.

5.16.1 Detailed Description

Structure with the shadow registers.

Definition at line 72 of file 16c550.h.

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

- mw/16c550.h

5.17 WfBuf Union Reference

Command buffer. Allows accessing the buffer as a command or as raw data.

`#include <cmds.h>`

Data Fields

- uint8_t data [WF_MAX_DATALEN]
  8-bit data
- uint16_t wdata [WF_MAX_DATALEN/2]
  16-bit data
- uint32_t dwdata [WF_MAX_DATALEN/4]
  32-bit data
- WfCmd cmd
  Command.

5.17.1 Detailed Description

Command buffer. Allows accessing the buffer as a command or as raw data.

Definition at line 67 of file cmds.h.

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

- cmds.h

5.18 WfCmd Struct Reference

Command definition.

`#include <cmds.h>`

Data Fields

- uint16_t cmd
  Command code.
- uint16_t len
  Command length.
- union {
  uint8_t data [WF_MAX_DATALEN - 4]
  8-bit data
  uint16_t wdata [(WF_MAX_DATALEN - 4)/2]
  16-bit data
  uint32_t dwdata [(WF_MAX_DATALEN - 4)/4]
  32-bit data
  WfMemRange mem
  Memory range.
  };

5.18.1 Detailed Description

Command definition.

Definition at line 51 of file cmds.h.

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

- cmds.h

5.19 WfMemRange Struct Reference

Memory range definition.

```
#include <cmds.h>
```

Data Fields

- uint32_t addr
  Start address of the range.
- uint32_t len
  Length of the memory range.

5.19.1 Detailed Description

Memory range definition.

Definition at line 45 of file cmds.h.

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

- cmds.h

Index