MooresCloud Holiday
EEPROM Data Structures
DRAFT 17th August 2013

The EEPROM storage on the Holiday is currently an On Semiconductor CAT24C64WI-GT3 64 kb (8kB) I2C CMOS Serial EEPROM. The device is rated at 1 million program/erase cycles, and 100 years data retention, when running off 5V and at 25°C ambient temperature. There is also some EEPROM available within the ATmega328, which may be used for other data.

The I2C bus is shared between the imx233 and ATmega328 processors. Normally the ATmega is the bus master at boot, until put into slave mode by the Linux environment via a serial command. The (7-bit) I2C address is 0b1010000 or 0x50. Erased bytes are 0xFF by default. When writing to the EEPROM, up to one page (typically 32 bytes, page aligned) can be written in a single operation. The EEPROM will not respond to requests while writing data.

This document details the data structures used to divide the LED pattern programs in the EEPROM into individual pages. The EEPROM structure is 8192 bytes divided into 256 pages of 32 bytes. Larger EEPROMs (if used in future, max 64kB) will be treated as having larger individual pages.

Pages 0 and 1 are used for control and program index, and pages 2 to 255 are for pattern programs. Programs use one or more pages each. There is no page allocation table, so adding new programs first requires reading (at a minimum) the first byte of a page to determine whether it is in use (0xFF = unused).

Page 0: Control data

- Byte 0 = Indicator byte = 'h' = 0x68 = 0b01101000
- Byte 1 = Layout version = 0x01 for this document
- Byte 2 = size of EEPROM in kB (typically 8, multiply this by 4 to get page size)
- Byte 3 = #globes attached – default is 50, max is 250
- Byte 4 = #programs – range of 0 to 32 (0x00 to 0x40)
- Byte 5 = Start up program (0 to #programs-1)
- Bytes 6 to 31 = reserved

Page 1: Program index

- Bytes 0 to 31 contain starting page number of program 0 thru 31

Pages 2 to 255: Pattern programs

- All instructions start at even addresses
- Most instructions are 2 bytes, an opcode and a single byte operand
- Some instructions are 4 bytes, and have an opcode with 3 data bytes as operand
- Opcode 0xFF (erased cell) is reserved as "EMPTY" – and if in the first byte of a page indicates the page is unused

On start up of ATmega, it will validate the control data, initialise registers, check the start up program number is less than #programs, look up the start page for that program in the program index, and then start executing instructions for that program. During operation of the program, the up/down buttons will modify the spinner value register. If the centre button is pressed, then the registers are reinitialised and the next program is started (with wrap around to program 0).

If a Linux application starts sending LED data, then the ATmega gives up control to the Linux environment. Linux applications can request the current running EEPROM program, and decide whether to take immediate control (maybe continuing that same program), or possibly wait for another button press.

Note also that the state of buttons held down at boot is saved for Linux applications to query. EEPROM based programs are not started until those buttons are released. The first LED will indicate boot status (red/blue) during the start up program (unless another program is begun), so effectively 1 less LED is available.

A basic set of instructions are implemented. Most are 2 bytes (opcode + operand), some are 4 bytes (opcode + 3 byte operand). All instructions are located at an even address. Program flow can fall through from one page to the next, or a jump instruction can move to a new page (e.g. if the next page is already in use). 4 byte instructions should not cross page boundaries, as the first data byte in the new page could be 0xFF – use a delay of 0ms length to pad if necessary. For all skip instructions, the next instruction must be 2 bytes, and would typically be a jump.

Some operational registers are pre defined, and can be accessed by index 0 to 15

- R0 is Index register = 1 byte
- R1-R3 is Colour register = 3 bytes, in G R B format per WS2812 LED
- R4 is Spinner register = 1 byte (updated by up/down buttons)
- R5 to R13 register = 1 byte each
- R14 is Argument register - passed by some jump/call instructions
- R15 is #globes (set on start of program, but can be changed!)

Where a globe index operand specified, it should be 0 to #globes-1, wraps around at end of string
Where a globe count operand specified, it should be 1 to #globes, or 0 for all globes

There are always 256 pages in the EEPROM, a minimum of 32 bytes each
Based on 8kB EEPROM, pagesize = 32, eeprommax = 0x1FFF
For a larger EEPROM, pagesize = (eepromsize in bytes)/256, eeprommax = (eeprom size in bytes)-1

Opcodes:

- 0x00 = Delay, operand is number of 10ms units to delay
- 0x01 = Load Colour register, 3 byte operand = 8 bits each for G R B values
- 0x02 = Load Colour register and set globe, 3 byte operand = 8 bits each for G R B values – uses and increments Index register
- 0x03 = Write to single globe, operand is globe index – sets and increments Index register
- 0x04 = Write to multiple sequential globes, operand is globe count – uses and increments Index register
- 0x05 = Write to multiple sequential globes based on bit mask, 3 byte operand, first byte upper nibble = # bits in mask, first byte lower nibble = # repetitions, second and third bytes are 16 bit mask (1=set globe, 0=skip globe, MSB first) – uses and increments globe index
-
- 0x08 = skip next instruction if reg A is equal to reg B, operand upper nibble = reg A, lower nibble = reg B
- 0x09 = skip next instruction if reg A is less than reg B (operand as above)
- 0x0A = skip next instruction if reg A is greater than reg B (operand as above)
- 0x0B = increment reg A, and skip next instruction if result equals reg B (operand as above)
- 0x0C = decrement reg A, and if result is 0 sets reg A to reg B, and skips next instruction
- 0x0F = Copy register A to register B, operand upper nibble is reg A, lower nibble reg B
-
- 0x10 to 0x0F = Load register, operand is value
- 0x20 to 0x2F = add value to register, operand is value (using 2's complement can also subtract)
- 0x30 to 0x3F = logical OR value to register, operand is value
- 0x40 to 0x4F = logical AND value to register, operand is value
- 0x50 to 0x5F = logical XOR value to register, operand is value
-
- 0x80 to 0x8F = compare register, skip next instruction if register equals operand value
- 0x90 to 0x9F = skip next instruction if register is less than operand value
- 0xA0 to 0xAF = skip next instruction if register is greater than operand value
- 0xB0 to 0xBF = increment register, and skip next instruction if result equals operand value
- 0xC0 to 0xCF = decrement register, and if result is 0 sets register to operand value, and skips next instruction
-
- 0xF0 = jump instruction in same page, operand = pointer (0x00 to pagesize-1, even values only)
- 0xF1 = jump to start of new page, operand is page number
- 0xF2 = jump to instruction in new page, 3 byte operand, first 2 bytes are pointer (0x0000 to eeprommax, even values only), third byte is saved in Argument register
- 0xF8 = call subroutine, 3 byte operand, first 2 bytes are pointer (0x0000 to eeprommax, even values only), third byte is saved in Argument register – only a single call level is possible
- 0xF9 = return from subroutine, operand is saved in Argument register
- 0xFF = EMPTY (also STOP)
- any undefined opcode = STOP