## **Adding Data to a Page Without Erasing Each Time**

Raspberry Pi Pico using a Winbond W25Q16JV
Jed Margolin   Test Data     7/4/2024

```
Erase
Flash Page: 4160
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

Bytes 0 to 31
Flash Page: 4160
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

Bytes 32 to 63
Flash Page: 4160
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

Bytes 64 to 95
Flash Page: 4160
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

Bytes 96 to 127
Flash Page: 4160
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

This is the code I wrote (and ran) on my Raspberry Pi Pico. It runs on a board I made which sends the data out through a uart port. A CH340 converts it to USB for a PC running AccessPort Terminal program.
`http://www.sudt.com/en/ap/`

```
void test10()
{
#define PAGE 4160    // Sector is 260
   uint32_t myflash_target_address;
   uint16_t n;
   unsigned char flash_buffer[256];

   myflash_target_address = FLASH_TARGET_OFFSET + (PAGE<<8);   // 256 bytes per Page;

   ints = save_and_disable_interrupts();    // Disable Interrupts

// Erase the sector
   uart_puts(UART_ID,"Erase");
   flash_range_erase(myflash_target_address, FLASH_SECTOR_SIZE);

// Show that it is erased
   read_page(PAGE, 256);        // Read Page and send it out through uart in Hex

// Make first buffer
   for(n=0; n<256; n++) {flash_buffer[n] = 0xFF;}  // Start with all 0xFF
   for(n=0; n<32; n++) {flash_buffer[n] = n;}      // first data address 0 t0 31

// write
   flash_range_program(myflash_target_address, flash_buffer, FLASH_PAGE_SIZE);

// Display the page
   uart_puts(UART_ID,"Bytes 0 to 31");
   read_page(PAGE, 256);        // Read Page and send it out through uart in Hex

// Make second buffer
   for(n=32; n<64; n++) {flash_buffer[n] = n;}  // second data is 32 to 63

// write
   flash_range_program(myflash_target_address, flash_buffer, FLASH_PAGE_SIZE);

// Display the page
   uart_puts(UART_ID,"Bytes 32 to 63");
   read_page(PAGE, 256);        // Read Page and send it out through uart in Hex

// Make third buffer
   for(n=64; n<96; n++) {flash_buffer[n] = n;}  // thirst data is 64 to  95

// write
   flash_range_program(myflash_target_address, flash_buffer, FLASH_PAGE_SIZE);

// Display the page
```

```
  uart_puts(UART_ID,"Bytes 64 to 95");
  read_page(PAGE, 256);        // Read Page and send it out through uart in Hex

// Make fourth buffer
  for(n=96; n<128; n++) {flash_buffer[n] = n;}  // thirst data is 64 to  95

// write
  flash_range_program(myflash_target_address, flash_buffer, FLASH_PAGE_SIZE);

// Display the page
  uart_puts(UART_ID,"Bytes 96 to 127");
  read_page(PAGE, 256);        // Read Page and send it out through uart in Hex

  restore_interrupts(ints);

  LED_break();
}
//==================================
void read_page(uint16_t page, uint16_t number)
{
// #define FLASH_TARGET_OFFSET (512 * 1024)  // start of my flash data
// FLASH_PAGE_SIZE is 256    // already defined

  uint32_t myflash_target;
  uint32_t myflash_read_address;
  char *ptr;
  unsigned int m, n;

  myflash_target = FLASH_TARGET_OFFSET + (page<<8);
  myflash_read_address = XIP_BASE + myflash_target;

  strcpy(line,"\nFlash Page: ");
  my_itoa(page, nstr);
  strcat(line, nstr);
  strcat(line,"\n");
  uart_puts(UART_ID, line);

// read the page
  ptr = (char *)myflash_read_address;

  m = 32;
  line[0] = 0;  // Start with NULL
  for(n=0; n<256; n++)
  {
    my_hextoa2(*ptr++, nstr);
    strcat(line,nstr);

    if(--m == 0)
    {
      strcat(line, "\n");
```

```
        uart_puts(UART_ID, line);
        m = 32;
        line[0] = 0;  // start new line
      }
      else  { strcat(line,"  "); }
   }
   strcat(line,"\n\n");
   uart_puts(UART_ID, line);

}
//==================================================
```