

**ECSE 4235: Embedded Systems II**

Example C Library - BCM2835

Christopher Ghattas, Alex George

Group 3 // 03/17/2024

1. **Member Roles**

Christopher Ghattas – Software Lead, Scribe

Alex George – Software Support, Github Expert, Scribe

1. **Code Involvement**

Christopher Ghattas – Helped write the gpioread and gpiowrite for the assembly. Helped to link all the files together.

Alex George – Helped write gpioread and gpiowrite for the assembly, and wrote the C file.

1. **Deliverables**
2. The write up should include complete descriptions and specifications for use and function of each component.

**Function “gpioWRITE”**

The `gpioWRITE` function builds upon the groundwork laid in our prior Linux assembly squarewave project, forming the backbone of this program's functionality. It operates with two parameters: the GPIO number and the desired state (either high or low), stored in `r0` and `r1`. Initially, the function calls the file descriptor by accessing `/dev/gpiomem`, ensuring subsequent read and write operations will work. Following this, the program then maps GPIO registers to memory, capturing and preserving their virtual memory address within `r5`. This step enables direct access to GPIO registers, allowing control over the Raspberry Pi’s GPIO pins.

Once the desired state (`r1`) is received, the function calculates the necessary shifts required to configure the GPIO pin's function register (GPFSEL) based on the pin number provided by the user in the C file. It then uses bitwise operations to clear the specified pin field in the GPFSEL register and switch it to output mode - this allows the pin to be written to. Depending on the user-supplied value (stored in `r9`), the code either activates or deactivates the GPIO pin. For this purpose, it accesses the appropriate register address for setting (GPSET0) or clearing (GPCLR0) the pin and applies the necessary bitwise operations to adjust the pin accordingly. Upon achieving the desired state, the code restores the stack pointer and other registers back to default values before giving control back to the calling function. This is to make sure that the function is ready for subsequent calls, increasing the program's reliability if the function needs to be reused multiple times.

The code for this specific section can be found in the Appendix with comments inline.

**Function “gpioREAD”**

The ‘gpioREAD’ function also builds upon the groundwork laid in our prior Linux assembly square wave project. Our codebase is very similar to the gpioWRITE function in terms of how the registers are set up and where memory is being mapped. The difference is what happens if the code recognizes a 1 or a 0. To read the gpio pin value, the code uses bitwise operations to determine whether a GPIO pin is high or low. It begins by shifting the base address by the value of the LEV0 offset, this accesses the level register associated with GPIO pins. Then, it performs a logical right shift operation (LSR) on the obtained register value (r8) by the specified GPIO pin number to get the status of the desired pin. Subsequently, a bitwise AND operation is applied between the result of the shift operation and the bitmask #1, which grabs the LSB to determine whether the GPIO pin is high (1) or low (0).

The code for this specific section can be found in the Appendix with comments inline.

**Function “select”**

Implementing select would be redundant in this case as the GPIO is already being “selected” when it is introduced as a parameter. Any further implementation would be redundant. Select is a part of the gpioREAD and gpioWRITE functions.

1. **Testplan Doc**

In order to test our functions we followed the steps below:

Case 1: Set GPIO values to 1 and 0 interchangeably, and using an LED to determine whether the GPIO is actually outputting a high or low value from the pin

Case 2: Change GPIO pin to test multiple pins to make sure that high and low signals are interchangeable on all GPIO pins.

Case 3: To test readGPIO, change GPIO values between 1 and 0 to make sure that the readGPIO is reading the correct GPIO values.

Case 4: Adding to test case 3, also trying this method on multiple GPIO pins to make sure the read function can work no matter which pin is being used.

**Appendix**

**Final gpioWRITE.s code**

| ***@ GPIO21 Related* .equ GPCLR0, 0x28 *@ clear register offset* .equ GPSET0, 0x1c *@ set register offset*   *@ Args for mmap* .equ OFFSET\_FILE\_DESCRP, 0 *@ file descriptor* .equ mem\_fd\_open, 3 .equ BLOCK\_SIZE, 4096 *@ Raspbian memory page* .equ ADDRESS\_ARG, 3 *@ device address*  *@ The following are defined in /usr/include/asm-generic/mman-common.h:* .equ MAP\_SHARED, 1 *@ share changes with other processes* .equ PROT\_RDWR, 0x3 *@ PROT\_READ(0x1)|PROT\_WRITE(0x2)*  *@ Constant program data* .section .rodata device:  .asciz "/dev/gpiomem"  .text .global gpioWRITE gpioWRITE:  push {lr}  push {r0-r12}  ldr r2, =Stack\_Pointer  mov r3, sp  str r3, [r2] *@ Saves current stack pointer to be used later in program*  mov r7, r0 *@ Store input parameters (GPIO pin number) in r7*  mov r9, r1 *@ Store input parameters (1 for on, 0 for off) in r9*  *@ Open /dev/gpiomem for read/write and syncing*  ldr r1, O\_RDWR\_O\_SYNC *@ flags for accessing device*  ldr r0, mem\_fd *@ address of /dev/gpiomem*  bl open *@ Call open system call to open the file*  mov r4, r0 *@ use r4 for file descriptor*    *@ Map the GPIO registers to a main memory location so we can access them* *@ mmap(addr[r0], length[r1], protection[r2], flags[r3], fd[r4])*  str r4, [sp, #OFFSET\_FILE\_DESCRP] *@ r4=/dev/gpiomem file descriptor*  mov r1, #BLOCK\_SIZE *@ r1=get 1 page of memory*  mov r2, #PROT\_RDWR *@ r2=read/write this memory*  mov r3, #MAP\_SHARED *@ r3=share with other processes*  mov r0, #mem\_fd\_open *@ address of /dev/gpiomem*  ldr r0, GPIO\_BASE *@ address of GPIO*  str r0, [sp, #ADDRESS\_ARG] *@ r0=location of GPIO*  bl mmap  mov r5, r0 *@ save the virtual memory address in r5*   mov r10, r7 *@ Copies GPIO pin number to r10*   mov r3, #4 *@ Moving a value of 4 into r3 to be used for shifting bits*   shiftLoop:  add r3, r3, #4 *@ Increment r3, by shift value*  add r10, r10, #10 *@ Increment r10 by 10 for bit positions*  cmp r10, #10 *@ Compare r10 with value of 10*  blt shiftLoop *@ Branches back to shiftLoop if less than 10*   mov r11, #0x3 *@ Set mask register r11 to 0x3*  mov r6, #0x2 *@ Set pin activator register to 0x2*    lsl r11, r11, r10 *@ Logical shift left by r10 value*   add r12, r10, r10 *@ Doubles value in r10 in order to use less lsl instructions*  lsl r11, r11, r12 *@ Logical shift left by double the r10 value, resulting in 2 standard shifts*   lsl r6, r6, r10 *@ Logical left shift by r10 value*  lsl r6, r6, r12 *@ Logical shift left by double the r10 value, resulting in 2 standard shifts*  *@ Set up the GPIO pin function register in programming memory*  add r0, r5, r3 *@#GPFSEL2 @ calculate address for GPFSEL2*  ldr r2, [r0] *@ get entire GPFSEL2 register*  bic r2, r2, r11 *@#GPFSEL2\_GPIO22\_MASK@ clear pin field for GPIO22*  orr r2, r2, r6 *@#MAKE\_GPIO22\_OUTPUT @ enter function code for GPIO22*  str r2, [r0] *@ update register*   mov r10, #1 *@ Sets r10 to a value of 1 to compare with r9 whether it is on or off*  cmp r10, r9 *@ Compares r10 with r9*  beq onloop *@ if r9, user input, is equal to 1, turns on GPIO*  b offloop *@ if r9, user input, is equal to 0, turns off GPIO*   onloop:  *@ Turn GPIO22 on*   add r0, r5, #GPSET0 *@ calc GPSET0 address*   mov r3, #1 *@ turns on bit*  lsl r3, r3, r7 *@ shift bit to pin position*  orr r2, r2, r3 *@ sets bit*  str r2, [r0] *@ update register*   b end  offloop:  *@Turn GPIO22 off*  add r0, r5, #GPCLR0 *@ calc GPSET0 address*   mov r3, #1 *@ turns off bit*  lsl r3, r3, r7 *@ shift bit to pin position*  orr r2, r2, r3 *@ sets bit*  str r2, [r0] *@ update register*  end:  mov r0, #0  ldr r12, =Stack\_Pointer  ldr sp, [r12] *@ Restores the stack pointer*  pop {r0-r12}  pop {lr}  bx lr  GPIO\_BASE:  .word 0xfe200000 *@ GPIO Base address Raspberry Pi 4* mem\_fd:  .word device O\_RDWR\_O\_SYNC:  .word 2|256 @ O\_RDWR (2)|O\_SYNC (256) .data Stack\_Pointer:  .word 0** |
| --- |

**gpioREAD code**

| @ GPIO21 Related .equ GPCLR0, 0x28 @ clear register offset .equ GPSET0, 0x1c @ set register offset .equ GPLEV0, 0x34 @ level register offset  @ Args for mmap .equ OFFSET\_FILE\_DESCRP, 0 @ file descriptor .equ mem\_fd\_open, 3 .equ BLOCK\_SIZE, 4096 @ Raspbian memory page .equ ADDRESS\_ARG, 3 @ device address  @ The following are defined in /usr/include/asm-generic/mman-common.h: .equ MAP\_SHARED, 1 @ share changes with other processes .equ PROT\_RDWR, 0x3 @ PROT\_READ(0x1)|PROT\_WRITE(0x2)  @ Constant program data .section .rodata device:  .asciz "/dev/gpiomem"  .text .global gpioREAD gpioREAD:  push {lr}  push {r0-r12}  ldr r2, =Stack\_Pointer  mov r3, sp  str r3, [r2]  mov r7, r0  @ Open /dev/gpiomem for read/write and syncing  ldr r1, O\_RDWR\_O\_SYNC @ flags for accessing device  ldr r0, mem\_fd @ address of /dev/gpiomem  bl open  mov r4, r0 @ use r4 for file descriptor    @ Map the GPIO registers to a main memory location so we can access them @ mmap(addr[r0], length[r1], protection[r2], flags[r3], fd[r4])  str r4, [sp, #OFFSET\_FILE\_DESCRP] @ r4=/dev/gpiomem file descriptor  mov r1, #BLOCK\_SIZE @ r1=get 1 page of memory  mov r2, #PROT\_RDWR @ r2=read/write this memory  mov r3, #MAP\_SHARED @ r3=share with other processes  mov r0, #mem\_fd\_open @ address of /dev/gpiomem  ldr r0, GPIO\_BASE @ address of GPIO  str r0, [sp, #ADDRESS\_ARG] @ r0=location of GPIO  bl mmap  mov r5, r0 @ save the virtual memory address in r5   mov r10, r7   mov r3, #4   shiftLoop:  add r3, r3, #4  add r10, r10, #10  cmp r10, #10  blt shiftLoop   mov r11, #0x3  mov r6, #0x2    lsl r11, r11, r10   add r12, r10, r10  lsl r11, r11, r12    lsl r6, r6, r10   lsl r6, r6, r12   @ Set up the GPIO pin function register in programming memory  add r0, r5, r3 @#GPFSEL2 @ calculate address for GPFSEL2  ldr r2, [r0] @ get entire GPFSEL2 register  bic r2, r2, r11 @#GPFSEL2\_GPIO22\_MASK@ clear pin field for GPIO22  orr r2, r2, r6 @#MAKE\_GPIO22\_INPUT @ enter function code for GPIO22  str r2, [r0] @ update register    @ Read GPIO pin value  ldr r0,[r5, #GPLEV0]  lsr r2, r2, r7  and r7, r2, #1  mov r1, r2  @ add r0, r5, #GPLEV0 @ calc GPLEV0 address  @ ldr r2, [r0] @ get entire GPLEV0 register  @ and r2, r2, r3  @ lsr r2, r2, r7 @ shift to get value of GPIO pin  @and r2, r2, #1 @ mask to get only the GPIO pin's value   mov r0, #0  ldr r12, =Stack\_Pointer  ldr sp, [r12]   pop {r0-r12}  pop {lr}  bx lr  GPIO\_BASE:  .word 0xfe200000 @ GPIO Base address Raspberry Pi 4 mem\_fd:  .word device O\_RDWR\_O\_SYNC:  .word 2|256 @ O\_RDWR (2)|O\_SYNC (256) .data Stack\_Pointer:  .word 0  mov r2, #1 |
| --- |

Library.c Code

| #include "Library.h" #include <stdio.h>    int main() {  int a = 21*;*  int b = 0*;*  gpioWRITE(a,b);  printf("returned write\n")*;*   int chris = gpioREAD(a)*;*   printf("GPIO pin value: %d\n", chris)*;*  if (chris == 1) {  printf("GPIO PIN IS HIGH\n")*;*  }  else {  printf("GPIO PIN IS LOW\n")*;*  }    return(0)*;* } |
| --- |

Library.h

| #ifndef LIBRARY\_H #define LIBRARY\_H  //void GPIOselect(int GPIOnumber, int function)*;* void gpioWRITE(int GPIOnumber, int value)*;* int gpioREAD(int GPIOnumber)*;*  #endif |
| --- |

**References**

1. Website used for BCM2835 library - McCauley , M. (no date) *bcm2835*, *BCM2835: C library for broadcom BCM 2835 as used in Raspberry pi*. Available at: https://www.airspayce.com/mikem/bcm2835/index.html (Accessed: 24 January 2024).
2. Code reference from Group 8 from ECSE4230 for memory mapping information- <https://bob.cs.sonoma.edu/IntroCompOrg-RPi/sec-gpio-mem.html>
3. Website for raspberry pi peripherals - <https://datasheets.raspberrypi.com/bcm2711/bcm2711-peripherals.pdf>
4. Resource on how to read gpio pins - <https://forums.raspberrypi.com/viewtopic.php?t=49444>