-

ECSE 4235 Embedded Systems Spring 2024

Projects: Library Component : Delay

Group 2

Pedro Pedro-Cristobal and Chris Hernandez

02/19/24

Table of Contents:

Roles of Group Members:

Pedro Pedro Cristobal:
- Created Documentation
- Created Testing Plan and Implementation
- Created delay functions in C
- Tested Square Wave Output on C and Assembly

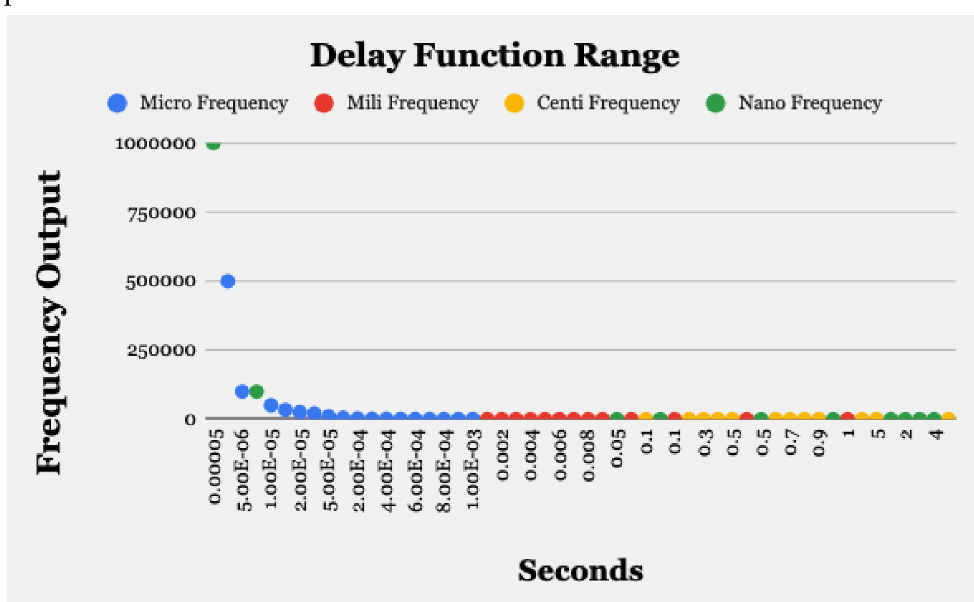
Chris Hernandez:
- Updated and Reviewed Documentation
- Assisted in creating and debugging delay functions
- Tested delay functions in C and Assembly

Tasks:

Overall Function Use:

There will be a total of six delay functions available: 4235_Delaynano, 4235_Delaymicro, 4235_Delaymilli, 4235_Delaycenti, 4235_Delaysec, and 4235_Delaymin. Each function corresponds to a specific time unit, ranging from nanoseconds to minutes. The user must be mindful of the appropriate function to use based on their desired time delay, as each function has its limitations. For instance, 4235_Delaymicro can cover a range from 1 to 100 microseconds, beyond which using milliseconds would be more suitable. The parameters for these functions are integers, meaning fractions of time units must be handled by combining multiple function calls. While certain functions like 4235_Delaysec can be used for longer time delays, users should understand the distinctions between the functions to select the most appropriate one for their application. The following functions are provided, and will state their parameters, outputs, examples calling from C and ARM8 Assembly, min and max inputs, and then a chart showing their accuracy to better show when to utilize other functions:

The following chart shows when to utilize the delay functions. As you can see many of the functions can be utilized outside their desired range, for example, nanoseconds can be used to output seconds, but it is all dependent on the user input. The chart shows which functions to use when desiring a certain time in seconds. For example, milliseconds covers a range from 0.001 seconds to 0.01 and centiseconds covers a range from 1, but evidently, with proper math calculations, centiseconds can be used to delay seconds. The same can be said for most of the functions, as in they can exceed their range starting from 1, but cannot get smaller, as decimal/ floating inputs cannot be utilized.



The frequency output from the chart references the testing plan, where the functions created were utilized in creating squarewaves with varying frequencies, and evidently, microseconds and nanoseconds output the higher frequencies.

initDelay:

void initDelay(int input)
Overview: This function sets the delay functions with respect with the RP4's current clock speed
Parameters: Integer value that represents clock speed ranging from 0 to 4294967295

Running Code in C:
 	
Extern (More Below):
extern void initDelay(int input);
 	
Call:
initDelay(input);

Running Code in Assembly: **Extern (More Below):** extern initDelay **Call:** ldr r0, =input bl initDelay
Return Type: Void

The following function is to be used alongside the E4235_WhatAmI(void) function where the function returns an integer number that should be stored and then placed as the input parameter of initDelay(input).

```
// Get clock Speed
int clock = E4235_WhatAmI();
printf("This is the clock: %i\n", clock);

// initialize the delay with clock
initDelay(clock);
```

E4235_Delaymin:

<table>
<tr><th>void E4235_Delaymin(int input)</th></tr>
<tr><td><u>Overview:</u>

This function stops and pauses the program for a given amount of minutes</td></tr>
<tr><td><u>Parameters:</u>

<u>Integer</u> value that user needs to delay the program ranging from 0 to 4294967295</td></tr>
<tr><td><u>Running Code in C:</u>

<u>Extern (More Below):</u>
extern void E4235_Delaymin(int input);

<u>Call:</u>
E4235_Delaymin(input);</td></tr>
<tr><td><u>Running Code in Assembly:</u>

<u>Extern (More Below):</u>
extern E4235_delaymin

<u>Call:</u>
ldr r0, =input
bl E4235_Delaymin</td></tr>
<tr><td><u>Return Type:</u>

Void</td></tr>
</table>

More:

- The following has to be done before using the function:
  - The delay functions file that contains the main program has to be within the same directory.
  - Using the keyword “extern” for the function which indicates that the function is declared elsewhere in a separate file. This allows for the

compiler to know about these functions without needing their implementation details at the current point of declaration, which ensures that the compiler knows their return type and parameter types.

- The following is the extern for a c file:
  - extern void E4235_Delaymin(int input);
- The following is the extern for a assembly file:
  - externE4235_Delaymin

E4235_Delaysec:

void E4235_Delaysec(int input)
Overview: This function stops and pauses the program for a given amount of seconds
Parameters: Integer value that user needs to delay the program ranging from 0 to 4294967295
Running Code in C: **Extern (More Below):** extern void E4235_Delaysec(int input); **Call:** E4235_Delaysec(input);
Running Code in Assembly: **Extern (More Below):** extern E4235_Delaysec **Call:** ldr r0, =input bl E4235_Delaysec
Return Type: Void

More:

- The following has to be done before using the function:
    - The delay functions file that contains the main program has to be within the same directory.

- Using the keyword “extern” for the function which indicates that the function is declared elsewhere in a separate file. This allows for the compiler to know about these functions without needing their implementation details at the current point of declaration, which ensures that the compiler knows their return type and parameter types.
- The following is the extern for a c file:
  - extern void E4235_Delaysec(int input);
- The following is the extern for a assembly file:
  - extern E4235_Delaysec

E4235_Delaycenti:

void E4235_Delaycenti(int input)
Overview: This function stops and pauses the program for a given amount of centiseconds
Parameters: Integer value that user needs to delay the program ranging from 0 to 4294967295
Running Code in C: **Extern (More Below):** extern void E4235_Delaycenti(int input); **Call:** E4235_Delaycenti(input);
Running Code in Assembly: **Extern (More Below):** extern E4235_Delaycenti **Call:** ldr r0, =input bl E4235_Delaycenti
Return Type: Void

More:

- The following has to be done before using the function:
    - The delay functions file that contains the main program has to be within the same directory.

- Using the keyword “extern” for the function which indicates that the function is declared elsewhere in a separate file. This allows for the compiler to know about these functions without needing their implementation details at the current point of declaration, which ensures that the compiler knows their return type and parameter types.
- The following is the extern for a c file:
  - extern void E4235_Delaycenti(int input);
- The following is the extern for a assembly file:
  - extern E4235_Delaycenti

E4235_Delaymilli:

void E4235_Delaymilli(int input)
Overview: This function stops and pauses the program for a given amount of milliseconds
Parameters: Integer value that user needs to delay the program ranging from 0 to 4294967295
Running Code in C: **Extern (More Below):** extern void E4235_Delaymilli(int input); **Call:** E4235_Delaymilli(input);
Running Code in Assembly: **Extern (More Below):** extern E4235_Delaymilli **Call:** ldr r0, =input bl E4235_Delaymilli
Return Type: Void

More:

- The following has to be done before using the function:
  - The delay functions file that contains the main program has to be within the same directory.

- Using the keyword “extern” for the function which indicates that the function is declared elsewhere in a separate file. This allows for the compiler to know about these functions without needing their implementation details at the current point of declaration, which ensures that the compiler knows their return type and parameter types.
- The following is the extern for a c file:
  - extern void E4235_Delaymilli(int input);
- The following is the extern for a assembly file:
  - extern E4235_Delaymilli

E4235_Delaymicro:

<table>
<tr><th>void E4235_Delaymicro(int input)</th></tr>
<tr><td><u>Overview:</u>

This function stops and pauses the program for a given amount of microseconds</td></tr>
<tr><td><u>Parameters:</u>

<u>Integer</u> value that user needs to delay the program ranging from 0 to 4294967295</td></tr>
<tr><td><u>Running Code in C:</u>

<u>Extern (More Below):</u>
extern void E4235_Delaymicro(int input);

<u>Call:</u>
E4235_Delaymicro(input);</td></tr>
<tr><td><u>Running Code in Assembly:</u>

<u>Extern (More Below):</u>
extern E4235_Delaymicro

<u>Call:</u>
ldr r0, =input
bl E4235_Delaymicro</td></tr>
<tr><td><u>Return Type:</u>

Void</td></tr>
</table>

More:

- The following has to be done before using the function:
  - The delay functions file that contains the main program has to be within the same directory.

- Using the keyword “extern” for the function which indicates that the function is declared elsewhere in a separate file. This allows for the compiler to know about these functions without needing their implementation details at the current point of declaration, which ensures that the compiler knows their return type and parameter types.
- The following is the extern for a c file:
  - extern void E4235_Delaymicro(int input);
- The following is the extern for a assembly file:
  - extern E4235_Delaymicro

E4235_Delaynano:

void E4235_Delaynano(int input)
Overview: This function stops and pauses the program for a given amount of nanoseconds
Parameters: Integer value that user needs to delay the program ranging from 0 to 4294967295
Running Code in C: **Extern (More Below):** extern void E4235_Delaynano(int input); **Call:** E4235_Delaynano(input);
Running Code in Assembly: **Extern (More Below):** extern E4235_Delaynano **Call:** ldr r0, =input bl E4235_Delaynano
Return Type: Void

More:

- The following has to be done before using the function:
  - The delay functions file that contains the main program has to be within the same directory.

- Using the keyword “extern” for the function which indicates that the function is declared elsewhere in a separate file. This allows for the compiler to know about these functions without needing their implementation details at the current point of declaration, which ensures that the compiler knows their return type and parameter types.
- The following is the extern for a c file:
  - extern void E4235_Delaynano(int input);
- The following is the extern for a assembly file:
  - extern E4235_Delaynano

Use of Functions:

- Timing and Synchronization:
  - Ensuring different parts of the system operate in harmony, meeting specific timing requirements.
    - Example: Clocks, Sensor Timing
- Debouncing Input Signals:
  - Eliminating false or erratic readings from switches or buttons by ignoring rapid state changes.
    - Examples: Switches, Buttons, Rotary Encoders
- PWM (Pulse Width Modulation):
  - Creating variable-width pulses for controlling motors, LEDs, or generating analog signals like audio.
    - Example: Generating a Square Wave with Various DutyCycles
- Implementing Protocols and Communication:
  - Ensuring accurate timing between data transmissions, essential for reliable communication.
    - Example: UART
- Software-Based Timers and Counters:
  - Facilitating tasks such as scheduling events, timeouts, or counting occurrences within the software.
- System Initialization and Stabilization:
  - Allowing time for components to settle or initialize properly before full operation begins, ensuring system stability.
- Testing and Debugging:
  - Introducing delays at specific points to observe behavior, diagnose timing-related issues, or slow down execution for easier debugging.
    - Example: Adding Delays in Critical Sections of Code for Observing Execution Flow During Debugging

Sample Code for C:

```
// include the header files needed for examples
#include <stdio.h>
#include "bcm2835.h"
// gcc -o pedro CMain.c E4235_Delays.s E4235_WhatAmI.s -l bcm2835
// extern each delay function for the compiler
extern void initDelay(int input);
extern int E4235_WhatAmI(void);
extern void E4235_Delaynano(int input);
extern void E4235_Delaymicro(int input);
extern void E4235_Delaymilli(int input);
extern void E4235_Delaycenti(int input);
extern void E4235_Delaysec(int input);
extern void E4235_Delaymin(int input);




// start of the main code
int main(int argc, char **argv)
{
   // Get clock Speed
   int clock = E4235_WhatAmI();
   printf("This is the clock: %i\n", clock);

   // initialize the delay with clock
   initDelay(clock);

   // user input for delay paramter
   int input = 50000;

   // initialze
   if (!bcm2835_init())
      return 1;

   // set gpio 22 as output
   bcm2835_gpio_fsel(2, BCM2835_GPIO_FSEL_OUTP);
   int seconds=0;
   // set up inifite loop
   while (1)
```

```
    {
       //printf("Amount of seconds: %i\n", seconds);
       // pull gpio high
       bcm2835_gpio_write(2, HIGH);

       // print statement
       //printf("This is the start:\n");

       // call sleep function
       // E4235_Delaymin(input);
       // E4235_Delaysec(input);
       // E4235_Delaycenti(input);
       // E4235_Delaymilli(input);
        E4235_Delaymicro(input);
       // E4235_Delaynano(input);

       // print statement
       //printf("This is the end:\n");

       // pull gpio LOW
       bcm2835_gpio_write(2, LOW);

       // call sleep function
       // E4235_Delaymin(input);
       // E4235_Delaysec(input);
       // E4235_Delaycenti(input);
       // E4235_Delaymin(input);
        E4235_Delaymicro(input);
       // E4235_Delaynano(input);
       //seconds++;
    }
    bcm2835_close();

} // main function
```

Assembly Example:

@ mmap part taken from by https://bob.cs.sonoma.edu/IntroCompOrg-RPi/sec-gpio-mem.html

```
@ Constants for blink at GPIO21
@ GPFSEL2 [Offset: 0x08] responsible for GPIO Pins 20 to 29
@ GPCLR0 [Offset: 0x28] responsible for GPIO Pins 0 to 31
@ GPSET0 [Offest: 0x1C] responsible for GPIO Pins 0 to 31

@ GPIO21 Related
.equ    GPFSEL2, 0x08   @ function register offset
.equ    GPCLR0, 0x28    @ clear register offset
.equ    GPSET0, 0x1c    @ set register offset
.equ    GPFSEL2_GPIO21_MASK, 0b111000000   @ Mask for fn register
.equ    MAKE_GPIO21_OUTPUT, 0b001000000     @ use pin for ouput
.equ    PIN, 22                    @ Used to set PIN high / low


@ 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

@ Misc
.equ    SLEEP_IN_S,1          @ sleep one second

@ 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"


@ The program start of code
    .text
```

```
@ extern all functions from the assembly file
    .extern initDelay
    .extern E4235_WhatAmI
    .extern E4235_Delaynano
    .extern E4235_Delaymicro
    .extern E4235_Delaymilli
    .extern E4235_Delaycenti
    .extern E4235_Delaysec
    .extern E4235_Delaymin

    .global main
.equ input, 1
main:
@ 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

@ Set up the GPIO pin funtion register in programming memory
    add     r0, r5, #GPFSEL2            @ calculate address for GPFSEL2
    ldr     r2, [r0]                    @ get entire GPFSEL2 register
    bic     r2, r2, #GPFSEL2_GPIO21_MASK@ clear pin field
    orr     r2, r2, #MAKE_GPIO21_OUTPUT @ enter function code
    str     r2, [r0]                    @ update register
```

```
    @ declare clock speed:
    ldr r0, =1800000000
    bl initDelay

loop:
    @ load in user input
    ldr r0, =input
    bl E4235_Delaysec
    @ turn on cdode
    add     r0, r5, #GPSET0 @ calc GPSET0 address
    mov     r3, #1          @ turn on bit
    lsl     r3, r3, #PIN    @ shift bit to pin position
    orr     r2, r2, r3      @ set bit
    str     r2, [r0]        @ update register

    @ load in user input
    ldr r0, =input
    bl E4235_Delaysec


    @ turn off code
    add     r0, r5, #GPCLR0 @ calc GPCLR address
    mov     r3, #1          @ turn on bits
    lsl     r3, r3, #PIN    @ shift bit to pin position
    orr     r2, r2, r3      @ set bit
    str     r2, [r0]        @ update register



b loop



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).
```

Test Plan:

Initializing Delay:

Since the RP4 clock frequency cannot be assumed to be fixed, a function called "initDelay" is created to ensure that the delay function takes into account any clock speed such as 1.5Ghz, 2.4Ghz, and any other future clock speeds. As mentioned above, the initDelay function takes in an integer number. Rather than using a fixed 1.8Ghz, two test cases were created to show that 1.5Ghz and 2.4Ghz can utilize the function. The actual functionality is explained within the commented code referenced at the reference and final code section:

Mathematical Relationship Established for Microseconds:

$$\frac{Raspberry\ Pi\ Clock\ Speed}{2(\ OnTime + Off\ Time)} = \frac{1{,}800{,}000{,}000}{2(\ OnTime + OffTime)} = \text{Frequency Desired}$$

Changing the Raspberry Pi Clock Speed, we can predict the behavior that will ensue if the clock speed is changed when setting the output to be a 1000 Hz square wave.

$$\frac{1{,}500{,}000{,}000}{2(900{,}000)} = 833.3\ \text{Hz}$$

$$\frac{1{,}800{,}000{,}000}{2(900{,}000)} = 1000\ \text{Hz}$$

$$\frac{2{,}400{,}000{,}000}{2(900{,}000)} = 1333\ \text{Hz}$$

Changing the Raspberry Pi Clock Speed, we can predict the behavior that will ensue if the clock speed is changed when setting the output to be a 5 Hz square wave.

$$\frac{1{,}500{,}000{,}000}{2(180{,}000{,}000)} = 4.16\ \text{Hz}$$

$$\frac{1{,}800{,}000{,}000}{2(180{,}000{,}000)} = 5\ \text{Hz}$$

$$\frac{2{,}400{,}000{,}000}{2(180{,}000{,}000)} = 6.6\ \text{Hz}$$

Code Used to Initialize:

The following program shows the initializing of the clock given the user input. This reflects on how the delay function is dependent on the RP4 clock speed.

```
initDelay:
        push {r1-r2}
        mov clockSpeed, r0                                  @ user input = clockSpeed
        mov r1,#2                                                   @ r1 = 2
        udiv clockSpeed, clockSpeed,r1          @ clock speed / 2
        ldr r1,=currentClock                            @ r1 = mem address of currentClock
        str clockSpeed,[r1]                                 @ store clock for access
        pop {r1-r2}
        bx lr
```

Creating Functions:

To implement the functions, loops were created to create our functions, where each function was independent from one another. To start, the seconds function was created to first output a delay of 1 second. By utilizing a simple On and Off code where a GPIO would be set ON and OFF, we can find the proper delay amount that would be necessary for one whole second.

Since there will be two delays, of 1 second each, the total period would be 2 seconds, which in theory should give us a 0.5 Hz output on an Oscilloscope. Given that the raspberry pi has clock speed of about 1800000000, we can utilize the following equation to obtain the initial value needed for one second:

Mathematical Relationship Established:

$$\frac{Raspberry\ Pi\ Clock\ Speed}{2(\ OnTime + Off\ Time)} = \frac{1{,}800{,}000{,}000}{2(\ OnTime + OffTime)} = \text{Frequency Desired}$$

$$\frac{1{,}800{,}000{,}000}{2(\ OnTime + OffTime)} = 0.5\text{ Hz}$$

$$\frac{1{,}800{,}000{,}000}{2(\ 2^{*}\ DelayInput\ )} = 0.5\ Hz$$

This would make DelayInput to be 900,000,000. Which is then divided by 2 in the initialization code mentioned above. After successfully outputting 0.5 Hz, additional code was utilized to account for user input. Here is a plan for one of the functions that will serve as the base and the only variable to be changed would be the delay input:

```
@ beginning of delayseconds Function
E4235_Delaymin:
        push {r0 - r1}                                  @ push registers r0 - r1

        ldr r2,=currentClock                    @ load current clock address
        ldr r3, [r2]                                    @ r3 = current clock
        ldr r4, =1                              @ r4 = 1000
        udiv r3, r3, r4                                 @ r3 = current clock / 1,000,000
        ldr r2,=calcClock                               @ load calc Clock to r2
        str r3,[r2]                                             @ store r3 for usage
        mov r5, #60
        mul r0, r0,r5
Min1:
        ldr r1, [r2]                    @ r1 = clock that was stored
Min2:
        subs r1, r1, #1                 @ r1 = r1 - 1, decrement r1
        bne Min2                                        @ repeat it until r1 = 0
        subs r0, r0, #1                 @ decrement minutes inputted
        bne Min1                                        @ reset R1
        pop {r0 - r1}                   @ pop registers r0 - r1
    bx lr                                       @ branch to LR value
```

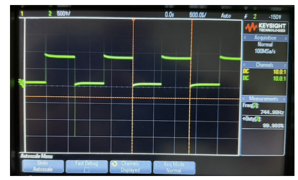
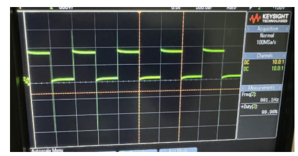
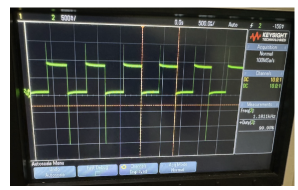
The following shows some test cases, but all of the test cases can be shown in the Testing Plan section:

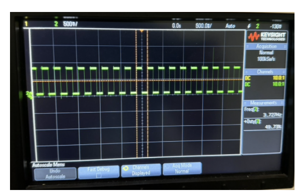
Delay Value Input	Frequency Programmed
1	500,000
5	100,000
10	50,000
15	33,333
20	25,000
25	20,000
50	10,000
.	.
.	.
.	.

Testing Plan:

Initializing the Clock:

As you can see from the below the expected frequency output given that the clock input changes shows that as the clock speed increases, the faster the functions output a square wave given the same parameters. The values tested were mentioned in the planning section.

Clock Speed:	Programmed Frequency	Frequency Output:	Picture Output
1.5Ghz	1,000 Hz	744.98 Hz	
1.8Ghz	1,000 Hz	991.1 Hz	
2.4Ghz	1,000 Hz	1181 Hz	

Clock Speed:	Programmed Frequency	Frequency Output:	Picture Output
1.5Ghz	5 Hz	3.7724 Hz	
1.8Ghz	5 Hz	4.999 Hz	
2.4Ghz	5 Hz	5.999Hz	

Seconds and Minutes:

To test seconds and minutes, a typical stopwatch is used where the program is run and the user will lap through after encountering a print statement on the terminal. Since both of the test codes for C and Assembly utilize an LED being put on and off, the user can also use that for reference to when to lap.

NOTE: Utilizing the oscilloscope is redundant since the Hz measured would be in decimal range and would take a longer time to check with larger values.

Nanoseconds, MicroSeconds, Milliseconds, and CentiSeconds:

To test the smaller delay functions, two test files were created which complete the following:

- Create an infinite Loop
- Call for a GPIO to be set on HIGH
- Insert Delay Function
- Call for GPIO to be set on LOW
- Insert Delay Function

This in turn creates a square wave where the calculated theoretical frequency can be found and then actually measured. The following shows how:

$$Frequency = 1 / Total\ Cycle\ Time = 1 / (2 * Delay\ Input) = Frequency\ in\ Hz$$

The following are the measured inputs and the outputs:

Chart For Nanoseconds:

Delay Value Input	Frequency Programmed	Frequency Outputted	Frequency Outputted Assembly
4000000000	0.125	0.112	0.1123
3000000000	0.1667	0.149	0.14980
2000000000	0.25	0.222	0.22466
1000000000	0.5	0.449	0.44943
900000000	0.55555	0.499	0.49999
500000000	1	.898	0.898
100000000	5	4.47	4.45

50000000	10	8.98	8.98
5000000	100,000	89.67	89.22
500000	1,000,000	885.49	889.

Chart for MicroSeconds:

Delay Value Input	Frequency Programmed	Frequency Outputted	Frequency Outputted Assembly
1	500,000	51.2	250000
5	100,000	38.1	100000
10	50,000	27.2	50000
15	33,333	27.2	33200
20	25,000	17.6	25000
25	20,000	15.1	20000
50	10,000	8.62	10080
100	5000	4650	5030
200	2500	2410	2516
300	1667	1637	1667.8
400	1250	1233	1258.18
500	1000	990	1000.002
600	833.33	827	838.3
700	714.29	710.6	716.7
800	625	622.2	624.5
900	555.56	553.4	555.8
1000	500	497	503.4

Chart for Milliseconds:

Delay Value Input	Frequency Programmed	Frequency Outputted	Frequency Outputted Assembly
1	500	501.50	500.000
2	250	252.34	250.020
3	166.7	166.13	166.74
4	125	125.42	125.01
5	100	101.21	100.03
6	83.3	83.34	83.34
7	71.4	71.24	71.44
8	62.5	63.2	62.502
9	55.6	55.4	55.556
10	50	50.5	50.005
100	5	5.4	5.005
500	1	1.2	0.99946
1000	0.5	0.6	0.49979
10000	0.05	0.062	0.0499999

Chart for Centi Seconds:

Delay Value Input	Frequency Programmed (Hz)	Frequency Outputted (Hz)	Frequency Outputted (Hz) Assembly
10	5	5.050	5.000001
20	2.5	2.53	2.50000
30	1.67	1.6895	1.66656
40	1.25	1.2668	1.250000
50	1	1.0140	0.999500

60	0.83	0.844	0.832500
70	.714	0.723.57	0.714000
80	0.625	0.620	0.624840
90	.556	0.56339	0.55501
100	0.5	0.506	0.49992
500	0.1	0.1023	0.099

References:

Assembly SquareWave:

- https://github.com/Herring-UGAECSE-4230-F23/Group-2/blob/main/Assembly/Linux%20ASM/Linux_asm_squarewave_final/Linux_asm_squarewave_final.s

RaspberryPi 4 Datasheet:

- https://datasheets.raspberrypi.com/bcm2711/bcm2711-peripherals.pdf

Extern Usage:

- https://developer.arm.com/documentation/dui0068/b/Directives-Reference/Miscellaneous-directives/EXTERN

Calling Assembly From C:

- https://developer.arm.com/documentation/dui0056/d/mixing-c--c----and-assembly-language/calling-between-c--c----and-arm-assembly-language/examples

WhatAmI Function:

- https://github.com/Herring-UGAECSE-4235/Class_Library_Components/tree/main/src

Final Programs:

E4235_Delay.s:

```
clockSpeed .req r12


@ start of file and global all functions to use in C and Asssembly Calling
        .text

   .text
        .cpu cortex-a7
        .global initDelay
        .global E4235_Delaymicro
        .global E4235_Delaymilli
        .global E4235_Delaycenti
        .global E4235_Delaysec
        .global E4235_Delaymin
        .global E4235_Delaynano

initDelay:
        push {r1-r2}
        mov clockSpeed, r0                          @ user input = clockSpeed
        mov r1,#2                                           @ r1 = 2
        udiv clockSpeed, clockSpeed,r1              @ clock speed / 2
        ldr r1,=currentClock                        @ r1 = mem address of currentClock
        str clockSpeed,[r1]                                 @ store clock for access
        pop {r1-r2}
        bx lr


@ beginning of delayseconds Function
E4235_Delaymin:
        push {r0 - r1}                      @ push registers r0 - r1

        ldr r2,=currentClock        @ load current clock address
        ldr r3, [r2]                        @ r3 = current clock
        ldr r4, =1                  @ r4 = 1000
        udiv r3, r3, r4                     @ r3 = current clock / 1,000,000
        ldr r2,=calcClock                   @ load calc Clock to r2
        str r3,[r2]                                 @ store r3 for usage
        mov r5, #60
        mul r0, r0,r5
Min1:
```

```
        ldr r1, [r2]            @ r1 = clock that was stored
Min2:
        subs r1, r1, #1         @ r1 = r1 – 1, decrement r1
        bne Min2                        @ repeat it until r1 = 0
        subs r0, r0, #1         @ decrement minutes inputted
        bne Min1                        @ reset R1
        pop {r0 - r1}           @ pop registers r0 - r1
   bx lr                        @ branch to LR value



@ beginning of delayseconds Function
E4235_Delaysec:
        push {r0 - r1}                  @ push registers r0 - r1

        ldr r2,=currentClock        @ load current clock address
        ldr r3, [r2]                    @ r3 = current clock
        ldr r4, =1                  @ r4 = 1000
        udiv r3, r3, r4                 @ r3 = current clock / 1,000,000
        ldr r2,=calcClock               @ load calc Clock to r2
        str r3,[r2]                             @ store r3 for usage

S1:
        ldr r1, [r2]            @ r1 = clock that was stored
S2:
        subs r1, r1, #1         @ r1 = r1 – 1, decrement r1
        bne S2                      @ repeat it until r1 = 0
        subs r0, r0, #1         @ decrement seconds inputted
        bne S1                      @ reset R1
        pop {r0 - r1}           @ pop registers r0 - r1
   bx lr                        @ branch to LR value


@ beginning of delaycenti Function
E4235_Delaycenti:
        push {r0 - r1}                  @ push registers r0 - r1

        ldr r2,=currentClock        @ load current clock address
        ldr r3, [r2]                    @ r3 = current clock
        ldr r4, =100                    @ r4 = 100
        udiv r3, r3, r4                 @ r3 = current clock / 1,000,000
        ldr r2,=calcClock               @ load calc Clock to r2
        str r3,[r2]                             @ store r3 for usage
```

```
C1:
        ldr r1, [r2]              @ r1 = clock that was stored
C2:
        subs r1, r1, #1           @ r1 = r1 – 1, decrement r1
        bne C2                            @ repeat it until r1 = 0
        subs r0, r0, #1           @ decrement centseconds inputted
        bne C1                            @ reset R1
        pop {r0 - r1}             @ pop registers r0 - r1
   bx lr                          @ branch to LR value


@ beginning of delaymilli Function
E4235_Delaymilli:
        push {r0 - r1}                            @ push registers r0 - r1

        ldr r2,=currentClock              @ load current clock address
        ldr r3, [r2]                              @ r3 = current clock
        ldr r4, =1000                             @ r4 = 1000
        udiv r3, r3, r4                           @ r3 = current clock / 1,000,000
        ldr r2,=calcClock                         @ load calc Clock to r2
        str r3,[r2]                                       @ store r3 for usage

M1:
        ldr r1, [r2]              @ r1 = clock that was stored
M2:
        subs r1, r1, #1           @ r1 = r1 – 1, decrement r1
        bne M2                                    @ repeat it until r1 = 0
        subs r0, r0, #1           @ decrement milliseconds inputted
        bne M1                                    @ reset R1
        pop {r0 - r1}             @ pop registers r0 - r1
   bx lr                          @ branch to LR value



@ beginning of delayMicro Function
E4235_Delaymicro:
        push {r0 - r1}                            @ push registers r0 - r1

        ldr r2,=currentClock              @ load current clock address
        ldr r3, [r2]                              @ r3 = current clock
        ldr r4, =1000000                          @ r4 = 1,000,000
        udiv r3, r3, r4                           @ r3 = current clock / 1,000,000
        ldr r2,=calcClock                         @ load calc Clock to r2
        str r3,[r2]                                       @ store r3 for usage
```

```
mm1:
        ldr r1, [r2]            @ r1 = clock that was stored
mm2:
        subs r1, r1, #1         @ r1 = r1 – 1, decrement r1
        bne mm2                                 @ repeat it until r1 = 0
        subs r0, r0, #1         @ decrement microSeconds inputted
        bne mm1                                 @ reset R1
        pop {r0 - r1}           @ pop registers r0 - r1
   bx lr                        @ branch to LR value

@ beginning of delayNano Function
E4235_Delaynano:
        push {r0}                       @ push registers r0
Nan:

        subs r0, #1                     @ decrement User Input
        bne Nan                                 @ repeat until r0 = 0 ( User input = 0)
   pop {r0}                     @ pop registers r0
        bx lr                           @ branch to LR value


.data
        currentClock:
                .word 0
        calcClock:
                .word 0
```

AMain.s:

This program is our implementation of BlinkLED in Assembly using DelayFunctions.s. The program takes a value as input which is stored in r0,. It is a parameter for the delay functions accessed globally as they use the extern calling convention. The delay functions are called by using a branch and link instruction (bl) followed by calling the function desired.

```
@ mmap part taken from by https://bob.cs.sonoma.edu/IntroCompOrg-RPi/sec-gpio-mem.html


@ Constants for blink at GPIO21
@ GPFSEL2 [Offset: 0x08] responsible for GPIO Pins 20 to 29
@ GPCLR0 [Offset: 0x28] responsible for GPIO Pins 0 to 31
@ GPSET0 [Offest: 0x1C] responsible for GPIO Pins 0 to 31

@ GPIO21 Related
.equ    GPFSEL2, 0x08   @ function register offset
.equ    GPCLR0, 0x28    @ clear register offset
.equ    GPSET0, 0x1c    @ set register offset
.equ    GPFSEL2_GPIO21_MASK, 0b111000000   @ Mask for fn register
.equ    MAKE_GPIO21_OUTPUT, 0b001000000     @ use pin for ouput
.equ    PIN, 22                         @ Used to set PIN high / low


@ 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

@ Misc
.equ    SLEEP_IN_S,1           @ sleep one second

@ 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"


@ The program start of code
```

```
    .text

@ extern all functions from the assembly file
    .extern initDelay
    .extern E4235_WhatAmI
    .extern E4235_Delaynano
    .extern E4235_Delaymicro
    .extern E4235_Delaymilli
    .extern E4235_Delaycenti
    .extern E4235_Delaysec
    .extern E4235_Delaymin

    .global main
.equ input, 1
main:
@ 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

@ Set up the GPIO pin funtion register in programming memory
    add     r0, r5, #GPFSEL2            @ calculate address for GPFSEL2
    ldr     r2, [r0]                    @ get entire GPFSEL2 register
    bic     r2, r2, #GPFSEL2_GPIO21_MASK@ clear pin field
    orr     r2, r2, #MAKE_GPIO21_OUTPUT @ enter function code
    str     r2, [r0]                    @ update register



    @ declare clock speed:
    ldr r0, =1800000000
```

```
    bl initDelay

loop:
    @ load in user input
    ldr r0, =input
    bl E4235_Delaysec
    @ turn on cdode
    add     r0, r5, #GPSET0 @ calc GPSET0 address
    mov     r3, #1          @ turn on bit
    lsl     r3, r3, #PIN    @ shift bit to pin position
    orr     r2, r2, r3      @ set bit
    str     r2, [r0]        @ update register

    @ load in user input
    ldr r0, =input
    bl E4235_Delaysec


    @ turn off code
    add     r0, r5, #GPCLR0 @ calc GPCLR address
    mov     r3, #1          @ turn on bits
    lsl     r3, r3, #PIN    @ shift bit to pin position
    orr     r2, r2, r3      @ set bit
    str     r2, [r0]        @ update register



b loop


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).
```

CMain.c:

This program is our implementation of BlinkLED in C using DelayFunctions.s and the bcm2835 library. The program takes an integer as input and is a parameter for the delay functions accessed globally as they use the extern calling convention. The delay functions are then called by function name with the delay as a parameter.

```
// include the header files needed for examples
#include <stdio.h>
#include "bcm2835.h"
// gcc -o pedro CMain.c E4235_Delays.s E4235_WhatAmI.s -l bcm2835
// extern each delay function for the compiler
extern void initDelay(int input);
extern int E4235_WhatAmI(void);
extern void E4235_Delaynano(int input);
extern void E4235_Delaymicro(int input);
extern void E4235_Delaymilli(int input);
extern void E4235_Delaycenti(int input);
extern void E4235_Delaysec(int input);
extern void E4235_Delaymin(int input);




// start of the main code
int main(int argc, char **argv)
{
    // Get clock Speed
    int clock = E4235_WhatAmI();
    printf("This is the clock: %i\n", clock);

    // initialize the delay with clock
    initDelay(clock);

    // user input for delay paramter
    int input = 50000;

    // initialze
    if (!bcm2835_init())
        return 1;

    // set gpio 22 as output
    bcm2835_gpio_fsel(2, BCM2835_GPIO_FSEL_OUTP);
    int seconds=0;
```

```
    // set up inifite loop
    while (1)
    {
        //printf("Amount of seconds: %i\n", seconds);
        // pull gpio high
        bcm2835_gpio_write(2, HIGH);

        // print statement
        //printf("This is the start:\n");

        // call sleep function
        // E4235_Delaymin(input);
        // E4235_Delaysec(input);
        // E4235_Delaycenti(input);
        // E4235_Delaymilli(input);
         E4235_Delaymicro(input);
        // E4235_Delaynano(input);

        // print statement
        //printf("This is the end:\n");

        // pull gpio LOW
        bcm2835_gpio_write(2, LOW);

        // call sleep function
        // E4235_Delaymin(input);
        // E4235_Delaysec(input);
        // E4235_Delaycenti(input);
        // E4235_Delaymin(input);
         E4235_Delaymicro(input);
        // E4235_Delaynano(input);
        //seconds++;
    }
    bcm2835_close();

} // main function
```