6.115 Laboratory 2: Timing and Controlling Events in the Physical World

David Ologan

June 9, 2022

1 Exercise 1: Check out more MINMON monitor code capability on your R31Jp board

- Add a "read" command to MINMON. At the MINMON prompt, you should be able to type a command that reads the hexi-decimal contents of a byte in external memory on the R31JP board. This hex byte should be printed on the PC screen.

Listing 1: Read MINMON Command

```
read:
   lcall getbyt         ; get address high byte
   mov dph, a           ; get the first half
   lcall prthex         ; print the first half
   lcall getbyt         ; get address low byte
   mov dpl, a           ; get the second half of the address
   lcall prthex         ; print the second half
   lcall crlf           ; get a newline
   movx a, @dptr        ; move entirety of dptr into a
   lcall prthex         ; print out the contents of the accumulator to terminal
   ljmp endloop                ; do jump by doing a ret
```

- Add a "write" command in MINMON. This command should load a hex byte into a specified location in external memory.

Listing 2: Write MINMON Command

```
write:
   lcall getbyt         ; get address high byte
   mov dph, a           ; get the first half
   lcall prthex         ; print the first half
   lcall getbyt         ; get address low byte
   mov dpl, a           ; get the second half of the address
   lcall prthex         ; print the second half
   lcall getchr
   lcall sndchr         ; get the equals sign
   lcall getbyt
   lcall prthex
   movx @dptr, a        ; move entirety of dptr into a
   lcall crlf           ; get a newline
   ljmp endloop         ; do jump by doing a ret
```

- You can test your new MINMON commands by first using the existing MINMON on your R31JP board to download your new test monitor code to RAM, and then executing the test code in RAM by flipping the MON/RUN switch. Test your code carefully. Read and write to several

locations in memory to make sure you understand the behavior of your code. Burn your finished monitor code into your code EEPROM using the 6.115 chip programmer you saw during the laboratory/kit familiarization lecture. You will not be able to use W to write to every location in external memory. BE prepared to explain why at your check-off.

You cannot write to every location in external memory because certain locations are reserved, an example of which is seen later in the lab. Additionally, you cannot overwrite addresses with MINMON since those aren't freely write able. .

2 Exercise 2: Reverse Assemble an Intel Hex file

- "Reverse assemble" the program. That is, read the hex file above, and write down the assembly language file that, assembled with AS31, would produce the above file.

Listing 3: Reverse Assembled Hex File

```
main:
mov A, #00h   ; Move 0 into the accumulator
mov 90h, 0E0h ; Copy data from location E0 to P1
setb 92h ; Set 2nd bit at location 90h (Port 1) to 1
sjmp main  ; Loop back to beginning
jnc 6Ch    ; Code never gets beyond this point, I'm assuming
orl c, 74h ; This is the secret message
mov R1, #20h
xrl A, R7
xrl A, R6
jb 69h, 82h
jb 36h, 45h
acall 45h
addc A, 21h
```

The secret message which begins after sjmp main, is in ASCII and reads : Party On 6.115!.

- Explain what the program does.

This code begins by moving 00h into the accumulator. Subsequently, it copies data from location E0h to 90h (Port 1). Then, it sets the 2nd bit of Port 1 to a 1, and loops back to the beginning. the code that follows is never reached because of the infinite loop.

3 Exercise 3: A Simple Calculator

- Write a program that reads two bytes from sequential locations in external memory, e.g. 9000h and 9001h. The program should compute the sum, difference, product, and the quotient of the two numbers. Store the results, i.e. sum, difference, 16-bit product, and quotient and remainder in sequential external memory locations without overwriting the original two bytes. Make the program easy to use repeatedly. It should be executed using the MINMON "G" command, not by toggling the MON/RUN switch. That is, the R31JP board should stay in MON mode (red LED lit) at all times. You should enter the operands 9000h and 9001h using your new "W" command. Complete the calculations using the "G" command, and read results from memory using your "R" command.

Listing 4: A Simple Calculator

```
.org 8000h
    ; Assumes the two numbers have been read in at 9000h and 9001h
    ; 9002h contains the addition
    ; 9003h contains the subtraction
```

```
    ; 9004h, 9005h contains the 16 bit multiplication
    ; 9006h, 9007h contains the quotient and remainder
main:
    mov dph, #90h
    mov dpl, #00h  ; Preset dph and dpl to 9000h
    movx A, @dptr  ; Move the value at 9000h into acc
    mov R1, A      ; First Number from #9000h into R1
    mov dpl, #01h  ; Update dpl to address of 2nd number
    movx A, @dptr  ; Move the value at 9001h into acc
    mov R2, A      ; Second Number from 9001h into R2
    lcall addition ; Call Addition
    lcall subtract ; Call Subtraction
    lcall multiply ; Call Multiplication
    lcall divide   ; Call Division
    ljmp 00h       ; Restart MINMON from beginning

addition:
    mov A, R1      ; Move first number from R1 into acc
    ADD A, R2      ; Add R2 to R1
    mov dpl, #02h  ; Update dpl to address for sum
    movx @dptr, A  ; Move sum into 9002h
    ret

subtract:
    mov A, R1      ; Move first number from R1 into acc
    SUBB A, R2     ; Subtract R2 from R1
    mov dpl, #03h  ; Update dpl to address for difference
    movx @dptr, A  ; Move difference into 9003h
    ret

multiply:
    mov A, R1      ; Move first number from R1 into acc
    mov B, R2      ; Move second number from R2 into B
    MUL AB         ; Multiply the two numbers together
    mov dpl, #05h  ; Update the address to hold the low byte of product
    movx @dptr, A  ; Move low byte of product into 9005h
    mov dpl, #04h  ; Update the address to hold the high byte of product
    mov A, B       ; Move high byte to acc
    movx @dptr, A  ; Move high byte of product into 9006h
    ret

divide:
    mov A, R1      ; Move first number from R1 into acc
    mov B, R2      ; Move second number from R2 into B
    DIV AB         ; Divide the first number by the second number
    mov dpl, #06h  ; Update the address to hold the quotient
    movx @dptr, A  ; Move quotient into 9006h
    mov dpl, #07h  ; Update the address to hold the remainder
    mov A, B       ; Move B into acc
    movx @dptr, A  ; Move remainder into 9007h
    ret
```

4 Exercise 4: Get Comfortable with Embedded Lighting Control

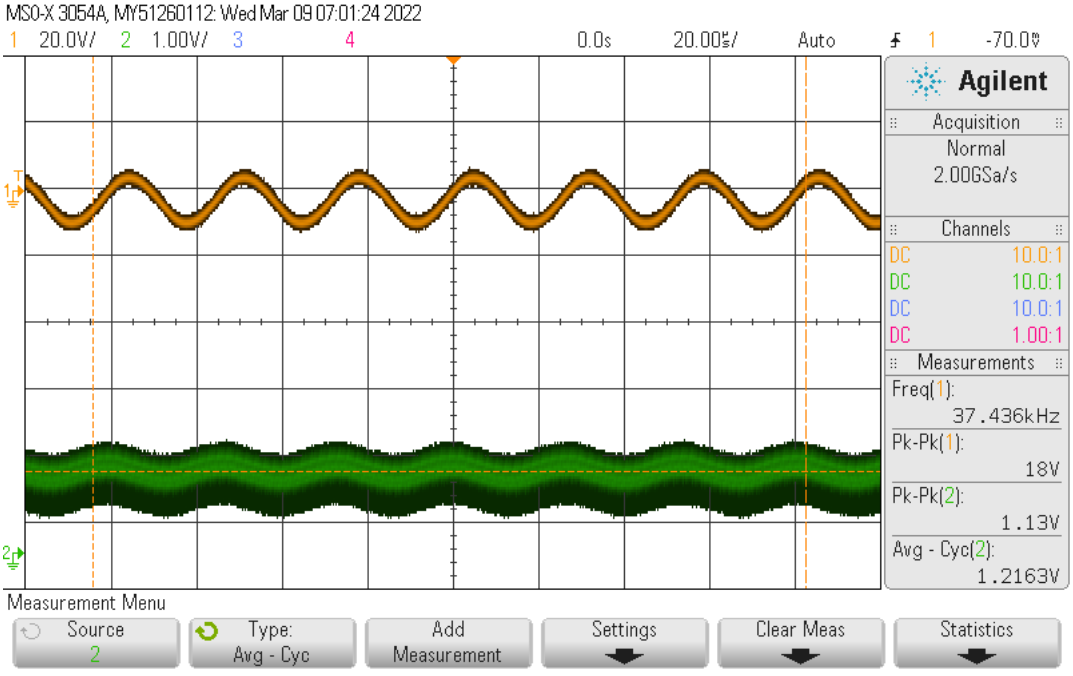
- Connect your LED PCB on a breadboard scrap as shown above. Using the INT setting on JP0 an your oscilloscope, study behavior of the circuit carefully. As you turn the blue POT for adjusting the ICM7555 drive frequency, what happens to the LED brightness? At what frequency are the LED's brightest? What would you predict the peak brightness frequency to be analytically? Do

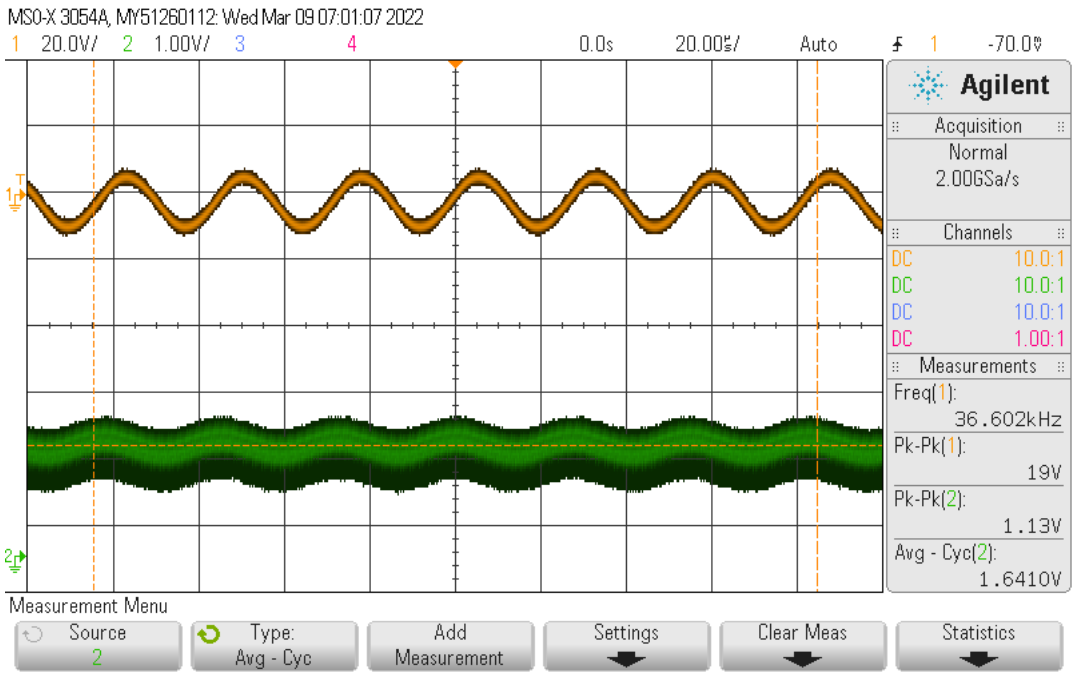
the experiment and the prediction agree?

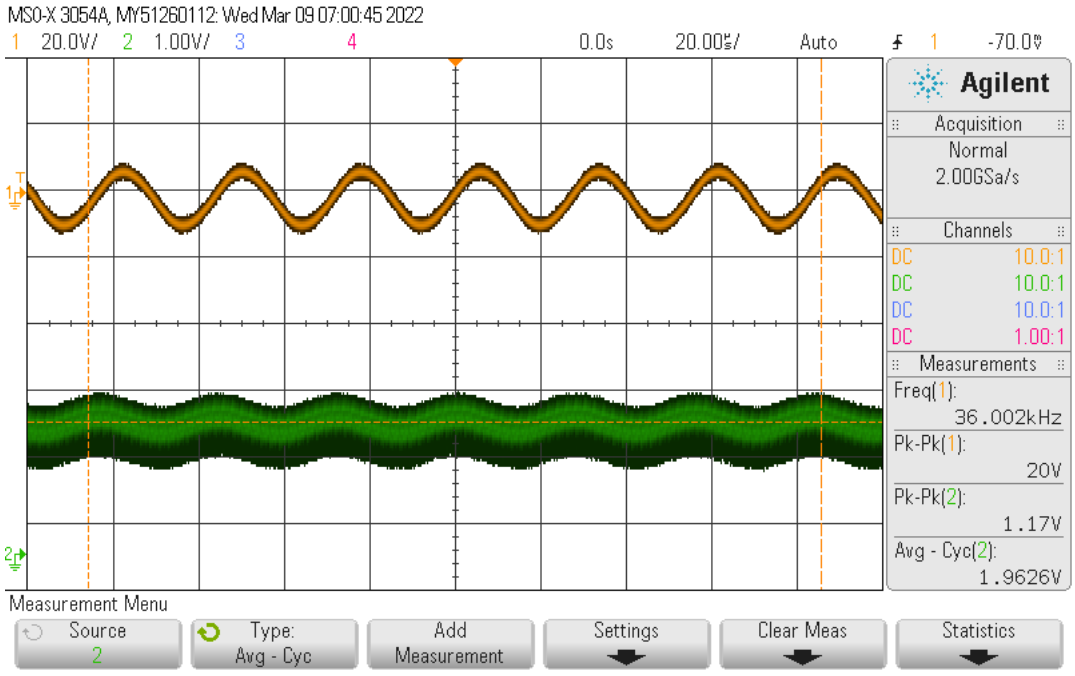
As you turn the blue POT for adjusting ICM7555 drive frequency, the LED brightness changes. The POT mimics a bell curve, where theres a peak at which the LED brightness is highest, but turning the POT in either direction from the peak causes the brightness to decrease. The frequency at which the LED's are brightest are 30kHz. Analytically, you would expect the brightness to be highest at $\frac{1}{2\pi(LC)}$. Basically the resonance point given by the LC in the circuit diagram.

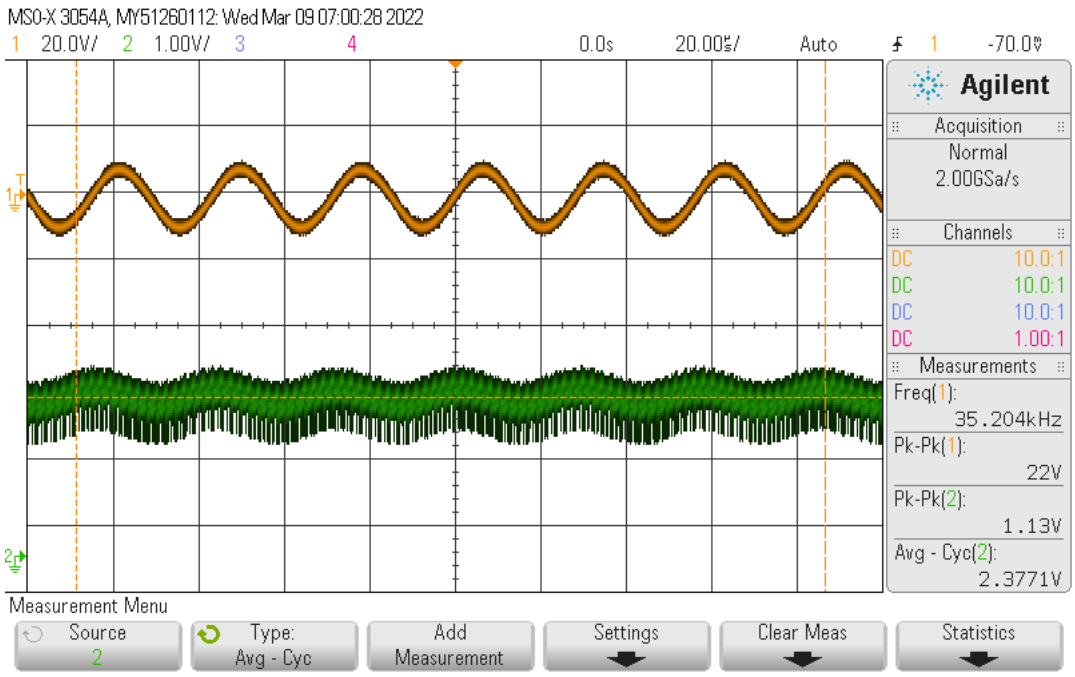
- Map the LED behavior as a function of frequency as you turn the POT. Make a table that includes ten levels of Vpk that correspond to ten different LED brightness levels. The lowest brightness level should be basically "off". At what frequency does this occur? The highest level should be the "brightest" you can get. At what frequency does this occur? Then, map other brightness levels and their associated frequencies, ranging smoothly from "off" to "brightest", for a total of ten entries in your table. Each table row should include frequency, Vpk, and qualitative brightness level as you assess it. The table has 10 rows. We will refer to it as the "brightness-frequency table".

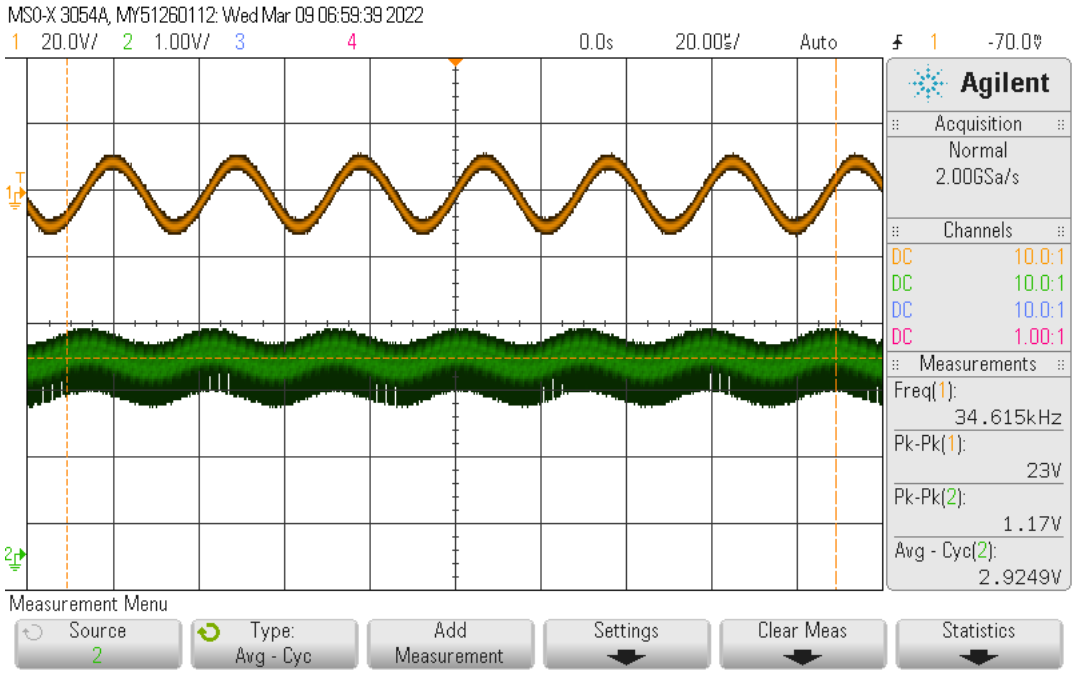
LED Behavior		
Frequency(kHz)	Vpk (V)	Qualitative Brightness Level
37.436 kHz	1.216 V	Basically Off
36.602 KHz	1.64 V	Barely On (Almost Unnoticeable)
36.002 kHz	1.96 V	Barely On (Noticeable)
35.204 kHz	2.37 V	Slight Glow
34.615 kHz	2.89 V	On but Dim
33.953 kHz	3.09 V	On
33.096 kHz	3.24 V	On (Medium)
32.222 kHz	3.37 V	On (Bright)
31.902 kHz	3.44 V	On (Bright)
30.486 kHz	3.49 V	Brightest Possible (Peak)

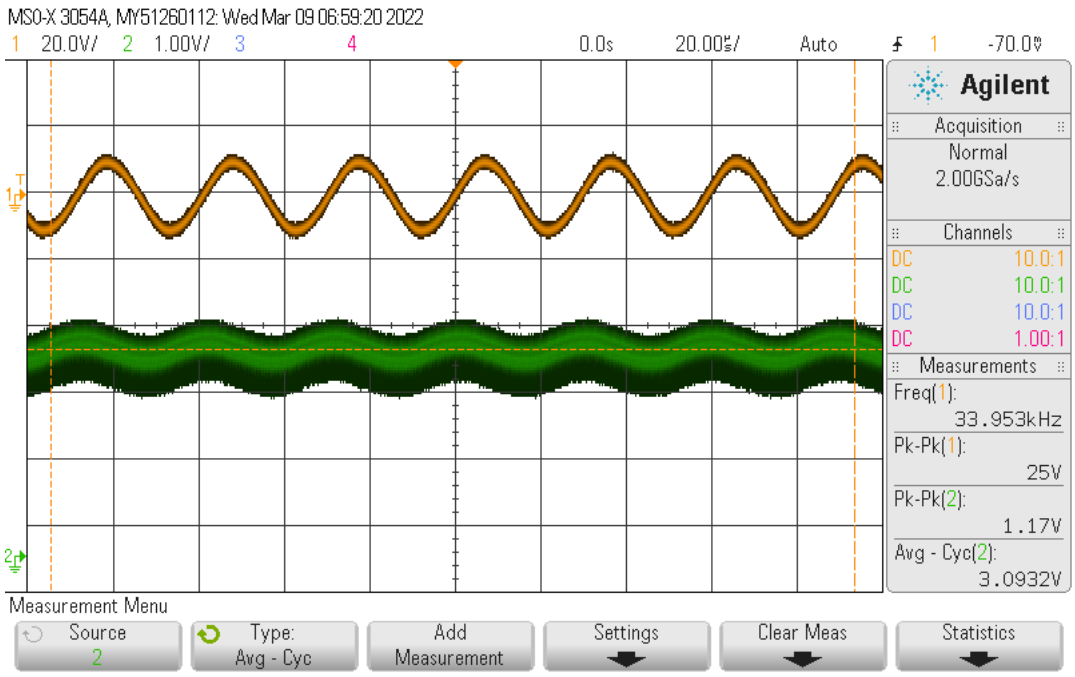


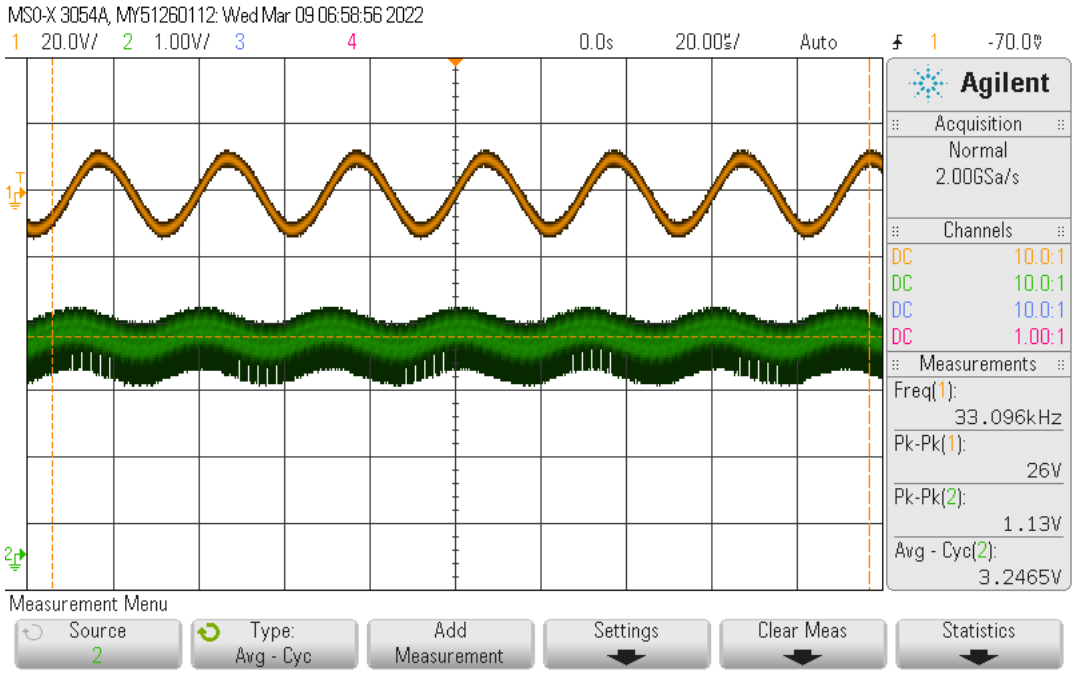

MSO-X 3054A, MY51260112: Wed Mar 09 07:01:07 2022
1 20.0V/ 2 1.00V/ 3 4 0.0s 20.00s/ Auto 1 -70.0
Agilent
Acquisition
Normal
2.00GSa/s
Channels
DC 10.0:1
DC 10.0:1
DC 10.0:1
DC 1.00:1
Measurements
Freq(1):
36.602kHz
Pk-Pk(1):
19V
Pk-Pk(2):
1.13V
Avg - Cyc(2):
1.6410V
Measurement Menu
Source 2
Type: Avg - Cyc
Add Measurement
Settings
Clear Meas
Statistics

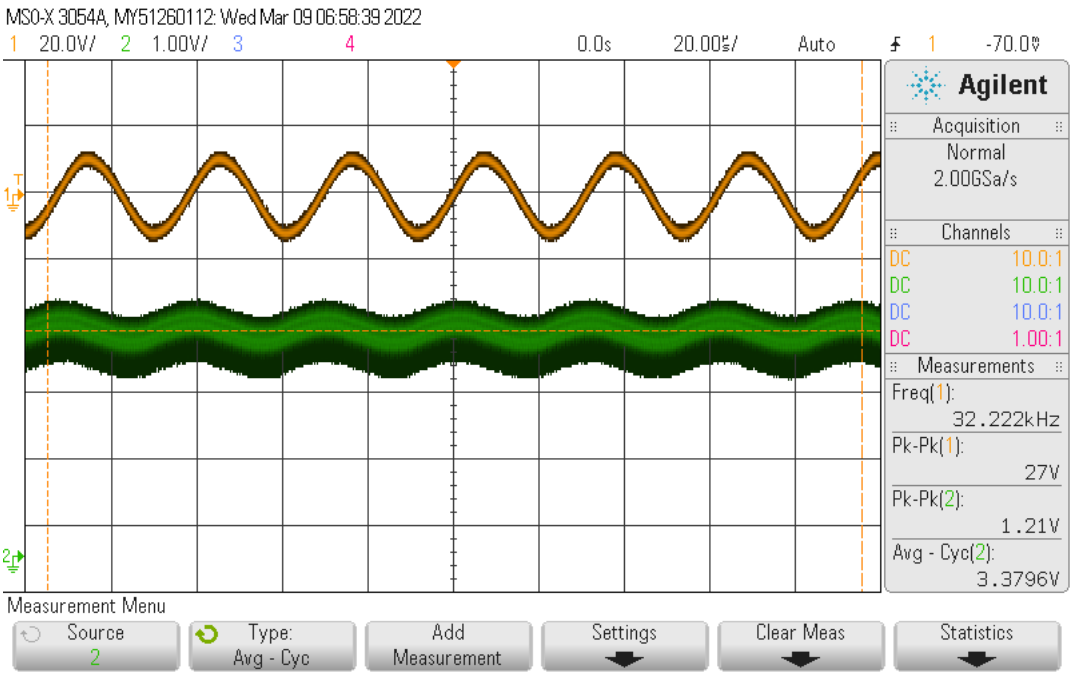

MSO-X 3054A, MY51260112: Wed Mar 09 07:00:45 2022
1 20.0V/ 2 1.00V/ 3 4 0.0s 20.00s/ Auto 1 -70.0
Agilent
Acquisition
Normal
2.00GSa/s
Channels
DC 10.0:1
DC 10.0:1
DC 10.0:1
DC 1.00:1
Measurements
Freq(1):
36.002kHz
Pk-Pk(1):
20V
Pk-Pk(2):
1.17V
Avg - Cyc(2):
1.9626V
Measurement Menu
Source 2
Type: Avg - Cyc
Add Measurement
Settings
Clear Meas
Statistics

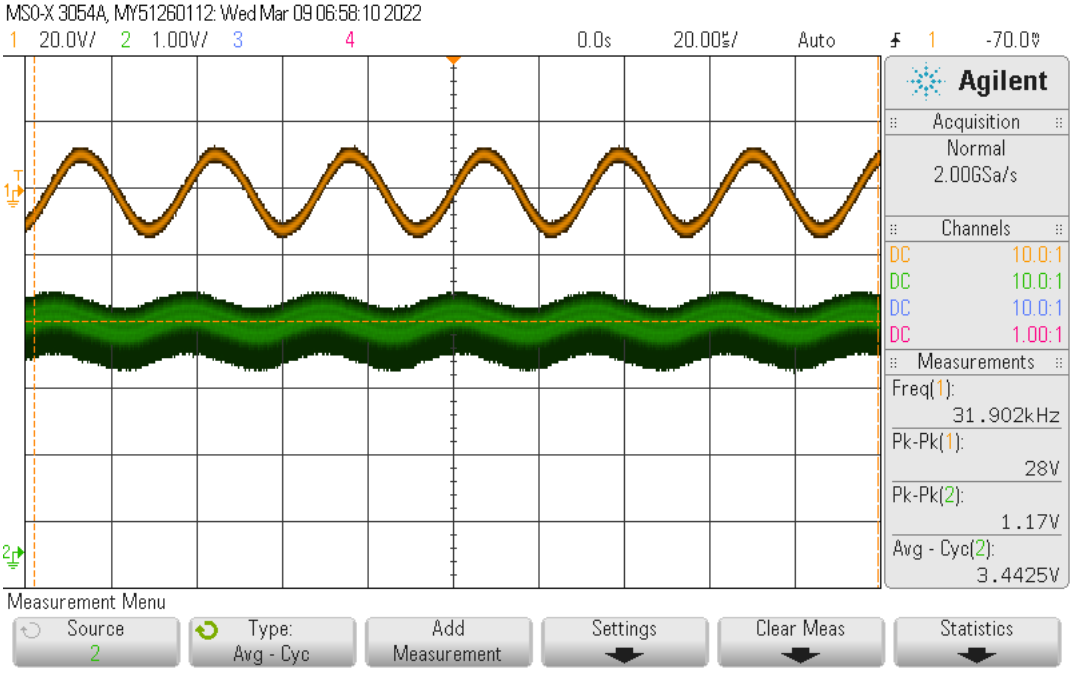

MSO-X 3054A, MY51260112: Wed Mar 09 07:00:28 2022
1 20.0V/ 2 1.00V/ 3 4 0.0s 20.00s/ Auto 1 -70.0
Agilent
Acquisition
Normal
2.00GSa/s
Channels
DC 10.0:1
DC 10.0:1
DC 10.0:1
DC 1.00:1
Measurements
Freq(1):
35.204kHz
Pk-Pk(1):
22V
Pk-Pk(2):
1.13V
Avg - Cyc(2):
2.3771V
Measurement Menu
Source 2
Type: Avg - Cyc
Add Measurement
Settings
Clear Meas
Statistics

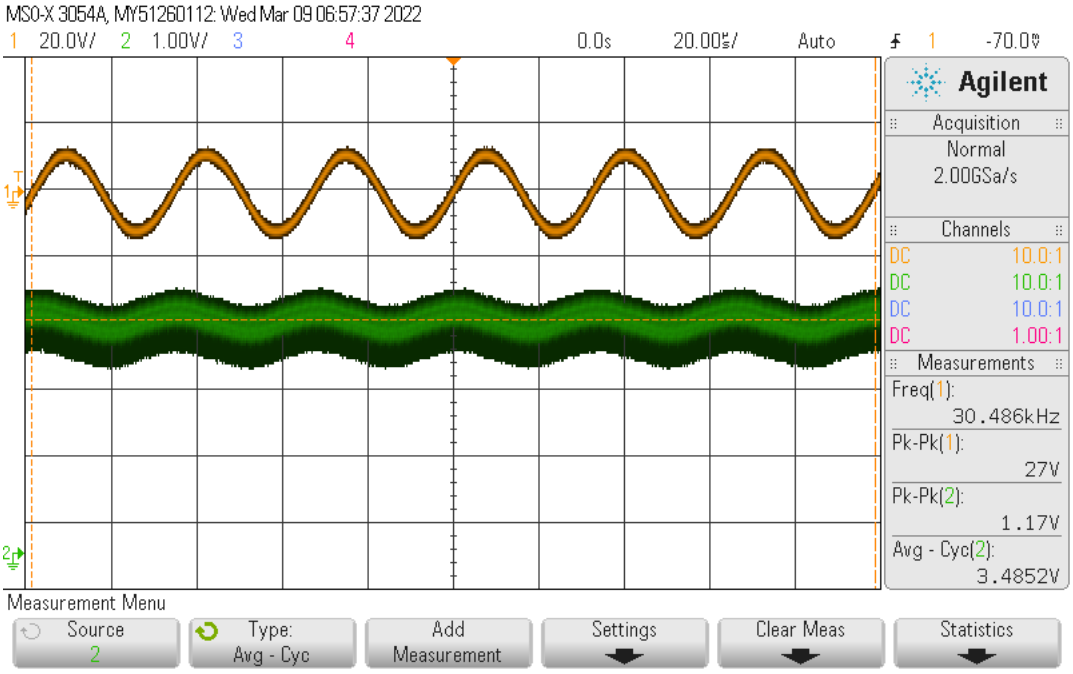

MSO-X 3054A, MY51260112: Wed Mar 09 06:59:39 2022
1 20.0V/ 2 1.00V/ 3 4 0.0s 20.00s/ Auto 1 -70.0
Agilent
Acquisition
Normal
2.00GSa/s
Channels
DC 10.0:1
DC 10.0:1
DC 10.0:1
DC 1.00:1
Measurements
Freq(1):
34.615kHz
Pk-Pk(1):
23V
Pk-Pk(2):
1.17V
Avg - Cyc(2):
2.9249V
Measurement Menu
Source 2
Type: Avg - Cyc
Add Measurement
Settings
Clear Meas
Statistics


MSO-X 3054A, MY51260112: Wed Mar 09 06:59:20 2022
1 20.0V/ 2 1.00V/ 3 4 0.0s 20.00s/ Auto 1 -70.0
Agilent
Acquisition
Normal
2.00GSa/s
Channels
DC 10.0:1
DC 10.0:1
DC 10.0:1
DC 1.00:1
Measurements
Freq(1):
33.953kHz
Pk-Pk(1):
25V
Pk-Pk(2):
1.17V
Avg - Cyc(2):
3.0932V
Measurement Menu
Source 2
Type: Avg - Cyc
Add Measurement
Settings
Clear Meas
Statistics


MSO-X 3054A, MY51260112: Wed Mar 09 06:58:56 2022
1 20.0V/ 2 1.00V/ 3 4 0.0s 20.00s/ Auto 1 -70.0
Agilent
Acquisition
Normal
2.00GSa/s
Channels
DC 10.0:1
DC 10.0:1
DC 10.0:1
DC 1.00:1
Measurements
Freq(1):
33.096kHz
Pk-Pk(1):
26V
Pk-Pk(2):
1.13V
Avg - Cyc(2):
3.2465V
Measurement Menu
Source 2
Type: Avg - Cyc
Add Measurement
Settings
Clear Meas
Statistics


MSO-X 3054A, MY51260112: Wed Mar 09 06:58:39 2022
1 20.0V/ 2 1.00V/ 3 4 0.0s 20.00s/ Auto 1 -70.0
Agilent
Acquisition
Normal
2.00GSa/s
Channels
DC 10.0:1
DC 10.0:1
DC 10.0:1
DC 1.00:1
Measurements
Freq(1):
32.222kHz
Pk-Pk(1):
27V
Pk-Pk(2):
1.21V
Avg - Cyc(2):
3.3796V
Measurement Menu
Source 2
Type: Avg - Cyc
Add Measurement
Settings
Clear Meas
Statistics


MSO-X 3054A, MY51260112: Wed Mar 09 06:58:10 2022
1 20.0V/ 2 1.00V/ 3 4 0.0s 20.00s/ Auto 1 -70.0
Agilent
Acquisition
Normal
2.00GSa/s
Channels
DC 10.0:1
DC 10.0:1
DC 10.0:1
DC 1.00:1
Measurements
Freq(1):
31.902kHz
Pk-Pk(1):
28V
Pk-Pk(2):
1.17V
Avg - Cyc(2):
3.4425V
Measurement Menu
Source 2
Type: Avg - Cyc
Add Measurement
Settings
Clear Meas
Statistics



5 Exercise 5: Make Some Square Waves

- Write a program that reads a byte from external memory. Count down from this number to zero. Every time you reach zero toggle the state of the Port 1 pin, then begin counting from the number to zero again. In this way you should be able to make a program that causes the micro controller to provide a variable frequency square wave output. You should not have to recompile the program to the frequency, i.e., the frequency should be controllable from MINMON. What is the highest frequency you can achieve? The lowest? What frequency resolution can you achieve, i.e, how finely can you adjust the frequency?

Listing 5: Square1

```
1  .org 8000h
2
3  main:
4      mov dph, #90h
5      mov dpl, #00h  ; Preset dph and dpl to 9000h
6      movx A, @dptr  ; Move the value at 9000h into acc
7      mov R0, A      ; Move acc into R0
8      loop:
9      DJNZ R0, loop  ; Decrement Loop until R0=0
10     cpl P1.0       ; Complement when Count over and restart
11     sjmp main
```

The maximum frequency I could achieve was around 66 kHz, the minimum frequency I could achieve was around 530 Hz.

- Repeat this exercise, but this time use a two-byte count instead of a single byte. You will be graded for elegance, i.e., use the smallest amount of code and register space you can to do this. How does the 16-bit count effect your highest achievable frequency? Lowest? Resolution? Again, make changes possible using MINMON so that the code does not have to be reassembled to change frequencies.

Listing 6: Square2

```
1  .org 8000h
```

```
main:
    mov dptr, #9001h
    movx A, @dptr  ; Move the value at 9001h into acc
    mov R0, A      ; R0, high byte counter
    mov dpl, #00h  ; Preset dph and dpl to 9000h
    movx A, @dptr  ; A, low byte counter
    loop:
    jz carry       ; jump if acc is 0
    return:
    dec A          ; Decrement low byte and continue on
    sjmp loop      ; restart

carry:
    CJNE R0, #00h, here   ; Checks if R0 is 0, if not go to here
    CJNE A, #00h, loop    ; Checks if high byte and low byte are 0, i.e count ends
    cpl P1.0              ; Complement P1.0
    sjmp main             ; Return to beginning to reset counters
    here:
    dec R0                ; Decrement high byte
    sjmp return           ; return to main loop
```

The 16 bit count lowers the maximum frequency that we could achieve. The maximum frequency I could achieve was around 54 kHz, the minimum frequency I could achieve was around 730 Hz.

- Write another program that generates variable frequency square waves. However, instead of using a count down loop, this time use the microcontroller's internal timer 0 configured in Mode 2, the auto re-load mode. Use a software timer interrupt to change the state of the Port 1 pin. Keep your code, particularly the interrupt service routine, as lean as possible to ensure fast execution. You will need to use the MON/RUN switch on your R31JP board to execute this program because interrupts are involved; that is, you cannot use the MINMON "G" command to run the program. Explain why. Nevertheless, if you are clever, you can make the frequency controllable by writing to a memory location using the MINMON "W" command before flipping the MON/RUN switch to RUN. This will avoid excessive assembling. What is the fastest frequency you can achieve? Slowest? Resolution?

Listing 7: Square/Interrupt

```
.org 00h
ljmp main

.org 00Bh

T0ISR:
    cpl P1.0       ; Interrupt Service Routine, Complement P1.0
    reti           ; reti to return to main loop

.org 0030h

main:
    mov dph, #10h
    mov dpl, #00h  ; Preset dph and dpl to 9000h
    movx A, @dptr  ; Move the value at 9000h into acc
    mov R0, A      ; Move value from acc into register

    mov TMOD, #02h ; Set Mode
    mov TH0, R0    ; Set Count
    setb TR0
    mov IE, #82h
    loop:          ; Infinite Loop
        sjmp loop
```

The maximum frequency I could achieve was around 55 kHz, the minimum frequency I could achieve was around 654 Hz.

- For the three pieces of code you just wrote (byte count, word count, and interrupt driven count), carefully analyze the timing of your code (number of machine cycles per instruction) and explain your observed highest and lowest frequencies and frequency resolution in terms of your code analysis.

The highest frequencies occurred with the first and third pieces of code. By minimizing word count and byte count, you decrease the required machine cycles and obtain a faster square wave. On the other hand, interrupts we slower than the first piece of code since we interrupt the main to service it, but functionally, this has many advantages since it minimizes our cost.

6 Exercise 6: Understand the 8254 Counter/Timer Chip

- The XIOSELECT pin is active (low) for access to memory addresses in the range FE00h through FEFFh. How many different address locations in this range select the control register on your 8254 chip? What 8254 register is located at FE00h? What 8254 registers are located at FE01h, FE02h, and FE03h, respectively?

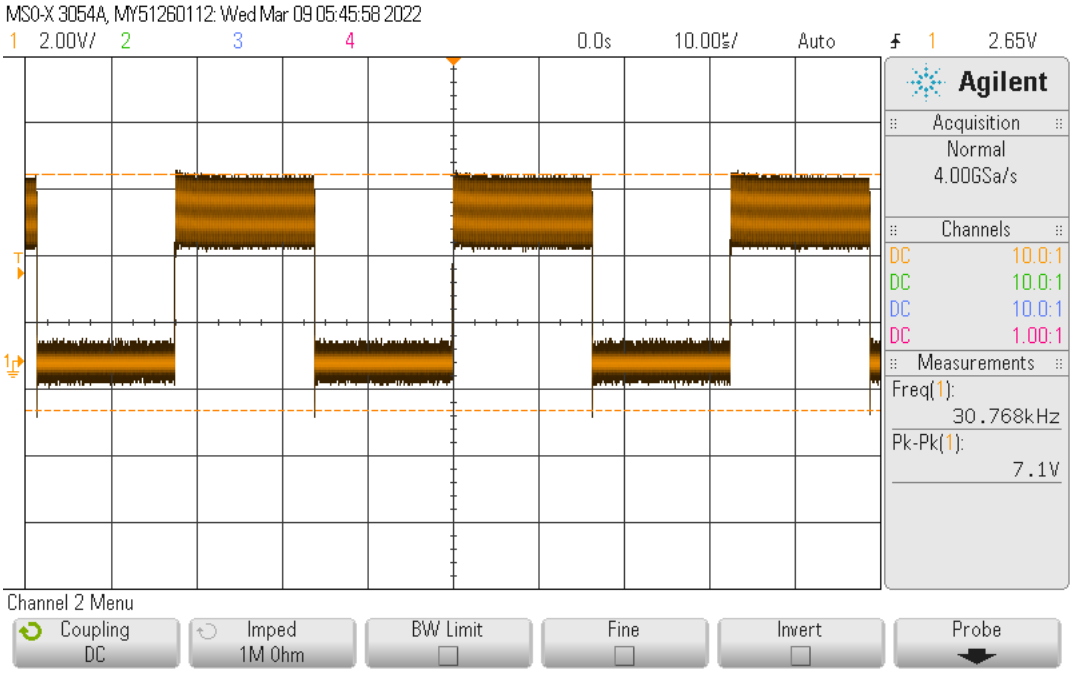
Four different address locations in this range select the control register on the 8254 Chip. FE00h references the Counter 0 register of the 8254. FE01h references Counter 1 and FE02h References Counter 2. FE03h is special in that it contains the control byte that allows you to set up the 8254, and use the counters you designate with the control word.

- For this question, imagine that we could drive the CLK0 pin either from a 1 MHz crystal or a 10 MHz crystal clock. Which choice gives the lowest possible 8254 output frequency on OUT0 in Mode 3? Which choice gives the highest possible output frequency? Which crystal provides the best output resolution around 30 kHz (where our LED light might operate)? That is, suppose the 8254 was operating in Mode 3 and producing an output square wave close to 30 kHz. Then suppose we change the "count" loaded into the timer register by adding or subtracting 1 from the count. Which input crystal, 1 MHz or 10 MHz, would provide the smallest change in output frequency?

Running the CLK0 pin with the 1 MHz crystal would give you the lowest possible output frequency. The 10 MHz crystal would give you the highest possible output frequency.The 10MHz crystal would also provide the best output solution around 30kHz. Since the 10MHz clock is faster, it gives you greater detail and control of your output frequency since the discrete time steps are smaller.

- For this part, connect the 8254 CLK0 pin to a 10 MHz TTL crystal oscillator. Use your MINMON read and write commands to configure the 8254 for Mode 3 operation. Produce a square wave as close to the peak brightness frequency of your LED lamp as you can.

The hex code that corresponds to the peak brightness frequency is 0145h.

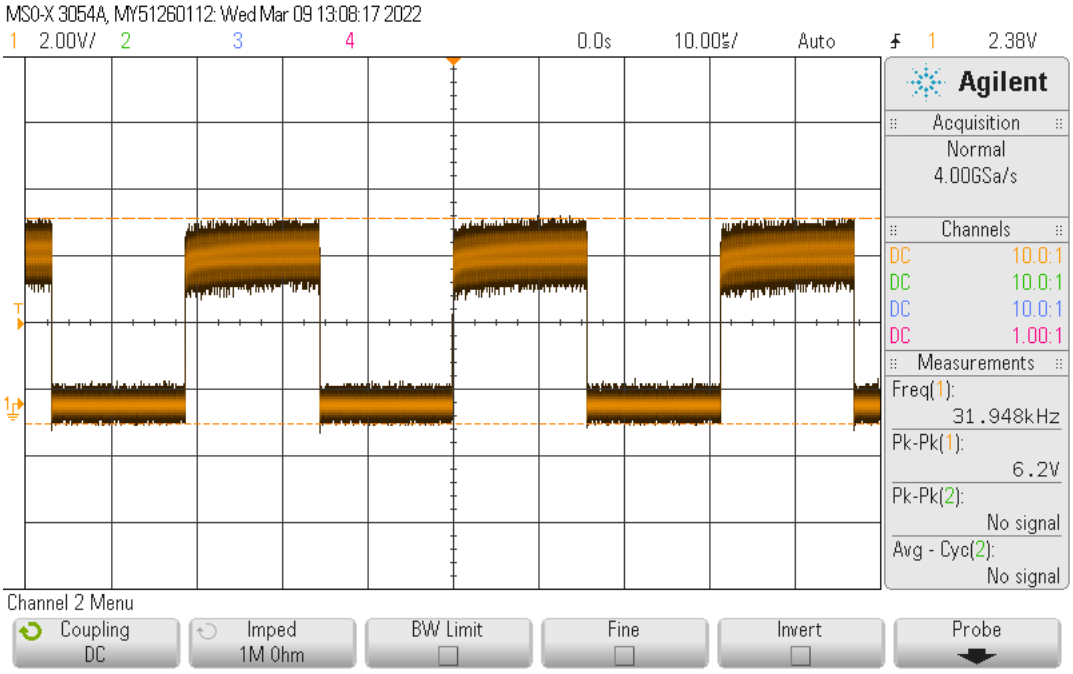


- Find the 8254 settings necessary to make each of the 10 frequencies in your brightness-frequency table. Expand the brightness-frequency table to include this valuable information for programming the 8254.

LED Behavior				
Keypad	Frequency(kHz)	Vpk (V)	Qualitative Brightness Level	8254 Hex Code
0	37.436 kHz	1.216 V	Basically Off	0109h
1	36.602 KHz	1.64 V	Barely On (Almost Unnoticeable)	0111h
2	36.002 kHz	1.96 V	Barely On (Noticeable)	0115h
3	35.204 kHz	2.37 V	Slight Glow	011Ch
4	34.615 kHz	2.89 V	On but Dim	0120h
5	33.953 kHz	3.09 V	On	0127h
6	33.096 kHz	3.24 V	On (Medium)	0130h
7	32.222 kHz	3.37 V	On (Bright)	0136h
8	31.902 kHz	3.44 V	On (Bright)	0139h
9	30.486 kHz	3.49 V	Brightest Possible (Peak)	0145h

7 Exercise 7: Control the LED lamp

- Use your 8254 to create the EXTernal square wave for the LED PCB. If you have made the connections indicated above, it should be easy to test your work. Use your MINMON "W" command to configure the 8254 to make a square wave corresponding to the "resonance" or maximum brightness of your LED PCB. With power on, you should see the LED's glow brightly. You can of course try other frequencies using MINMON and verify that the LED brightness changes appropriately.



- Now, let's make a lighting controller. This is a common, lucrative commercial product. You can see some examples here: https://www.lutron.com/en-US/Pages/Dimmers/Dimmers.aspx Here's the plan: write an R31JP program in assembly that uses your keypad and 74C922 (still available on Port 1) to control the light. When you press one of the "number" keys 0-9, the LEDs should go to one of 10 brightness levels. The "0" key should set the lowest level (essentially off), and the "9" key should set the brightest level. The intermediate keys should set monotonically increasing brightness levels ranging from the "0" key to the "9" key.

Listing 8: Brightness Levels

```
.org 00h
ljmp main

main:
    mov dptr, #0FE03h
    mov A, #36h
    movx @dptr, A    ; Update Control Byte with 36
    loop:
        lcall getkey      ; Get a Button Press
        lcall fix         ; Obtain the freq hex code in acc from database
        mov dptr, #0FE00h
        movx @dptr, A     ; Move in the low byte to #FE00h
        mov dptr, #0FE00h
        mov a, #01h
        movx @dptr, A     ; Move in the high byte to #FE00h
        sjmp loop

getkey:
    jnb P3.3, getkey      ; Jump if bit not set, wait for key press
    mov P1, #0FFh         ; Set Port 1 high to be read
    pressdone:
        jb P3.3, pressdone ; If press detected, wait for it to end
        mov a, P1         ; Reading to Port 1
        clr C
        SUBB A, #0F0h     ; Make first nibble 0s
        mov P1, a; Writing to Port 1
        ;clr P1
```

```
28          ret
29
30  fix:
31      inc a                   ; increment acc
32      movc a, @a+pc           ; get correct ASCII translation from data table
33      ret
34      .db 00h, 00h, 09h, 00h, 00h, 45h, 39h, 36h, 00h, 30h, 27h, 20h, 00h, 1Ch, 15h, 11h ;
            datatable with correct ASCII codes
```

- Now expand your program to add some of the other "professional features" you would find on something like the Lutron LED+ family of dimmers. When you press "A" on your keypad, your lights should ramp smoothly from "off" to "full bright" over a time period of about 10 seconds, and then remain on at full brightness. With the lights at "full brightness," when you press "B" on the keypad, your lights should ramp smoothly to "off" over a time period of about 10 seconds. Finally, again with the lights at "full brightness", when you press "C" on the keypad, your R31JP should keep track of how long the "C" key is held down, ranging between 0 to 10 seconds as decided by the user. Then, when the "C" key is released, the LEDs should remain "on" at full brightness over approximately the time period that the "C" key was held down, and then finally ramp to "off" over another period of about 10 seconds. These functions provide useful features, e.g., for "B" and "C," leaving you with some lighting as you walk out of a room, but then conserving energy by ramping the lights to off when the room is presumably empty. Or, with "A," providing a gradual increase in lighting as you enter to avoid a sudden blast of light to your eyes. Dimmers that provide these features are highly prized and command premium retail prices.

8 Exercise 8: Make Sure you Understand the 8051 Hardware

- How long (in seconds) is one machine cycle on this SBC:

One machine cycle on this SBC is about 1 uS. Given an 11.059 MHz clock and 12 operations per machine cycle, we know that the chip runs at around 1 uS per cycle.

- The memory subsystem contains 3 memory chips: Two ROMS labeled C1 and C3 in the schematic, and one RAM labeled D3 in the schematic. Which chip would most profitably contain MINMON?

C1 would most profitably contain MINMON, since its connected to Port 1 of the 74HC138.

- Given the SBC as connected in the attached schematics, after a power-on reset, the first code-store fetch will be from (circle all which apply):

The first code store fetch will be from Internal 8051 ROM. Since all the pins start high (active low), Internal ROM will be the first access point.

- The MINMON command "G8000" could be used to execute a program or routine on this SBC given only the chips and interconnect shown on the attached schematics:

True, it would reference D3.

- The MINMON command "G2000" would execute a program stored in (circle all which apply):

MINMON command "G2000" would execute a program stored in C3.

- Given the SBC as connected in the attached schematics, the 8051 instruction "MOVC A, @A+DPTR" could be used to read data bytes stored in (circle all which apply):

C1, C3, D3. This is because MOVC is used to read external memory ROM, which includes C1, and C3. It uses PSEN so D3 as well.

- Given the SBC as connected in the attached schematics, the 8051 instruction "MOVX A, @DPTR" could be used to read data bytes stored in (circle all which apply):

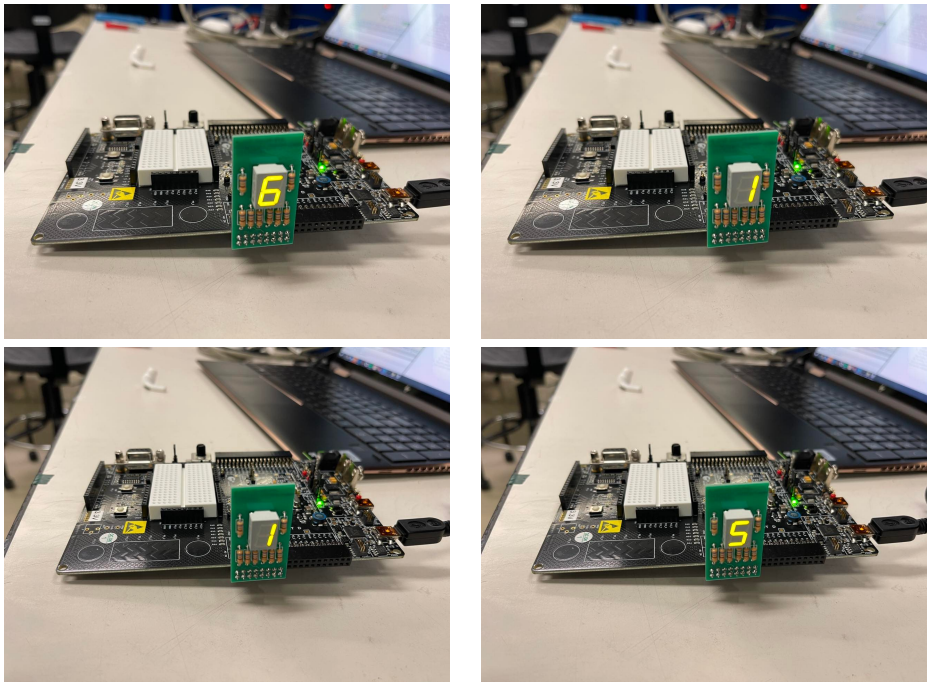
D3. This is because MOVx is used to read external RAM, which includes D3.

9 Exercise 9: Learn about the Cypress PSoC Big Board

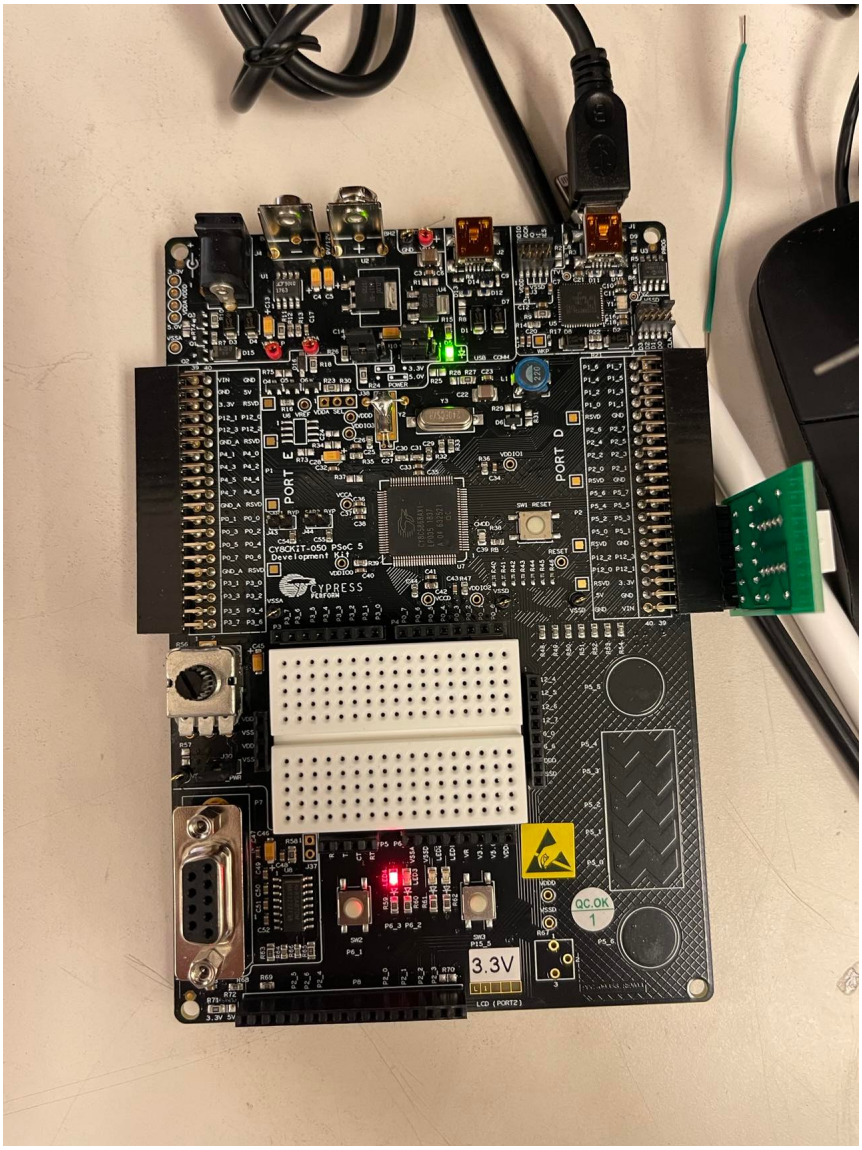
- Go to the "PSoC" page on the 6.115 course website: http://web.mit.edu/6.115/www/page/psoc-information.html

- Download the "Blinky" code for PSoC creator and make sure that you remember how to use Creator. Run Blinky so that you see the "6.115" message on your Digit LED display. In the Creator software, select the correct PSoC target device (CY8C5868AXI-LP035) using the "Device Selector..." under the "Project" menu tab. Use the "Build" menu tab to build Blinky, and use the "Debug" menu tab to program the PSoC evaluation board. Make sure you are comfortable "building" and "programming" the PSoC and that you can see the LED display working. In Creator, make sure that you can find and edit the files with ".cydwr", ".c", and ".cysch" for the project. These files are, respectively, the "design wide resources," the "Clanguage program code," and the internal "schematic" for the project.

- Now, from the 6.115 course website, download the "Exercise 1" for the PSoC (on the webpage right under the "Blinky" project). Complete Exercise 1 by following the instructions on the file page with a ".cysch" extension for Exercise 1. Build and program the project on your PSoC board. You should be rewarded with a regularly flashing light in your bank of "Groovy LEDs" on the evaluation board.



- Modify the c-code for the Exercise 1 project so that the light flashes a pattern corresponding to "SOS" in Morse Code: Three short flashes, followed by three long flashes, followed by three short flashes, followed by a 1 second pause. The SOS should repeat indefinitely. You may find the "CyDelay" command useful in your code.



- Save your modified project. Include a phone picture of the light flashing on, and also include your carefully commented, modified SOS C-code in your lab report.

Listing 9: SOS Code

```
/* ========================================
 *
 * Copyright YOUR COMPANY, THE YEAR
 * All Rights Reserved
 * UNPUBLISHED, LICENSED SOFTWARE.
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION
 * WHICH IS THE PROPERTY OF your company.
 *
 * This file is necessary for your project to build.
 * Please do not delete it.
 *
 * ========================================
*/

#include <device.h>

```

```
18  void main()
19  {
20
21      for(;;)
22      {
23          Pin_1_Write(~Pin_1_Read());    // write the value to Pin_1
24          CyDelay(250);                  // delay in milliseconds
25          Pin_1_Write(~Pin_1_Read());    // write the value to Pin_1
26          CyDelay(250);                  // delay in milliseconds
27          Pin_1_Write(~Pin_1_Read());    // write the value to Pin_1
28          CyDelay(250);                  // delay in
29          Pin_1_Write(~Pin_1_Read());    // write the value to Pin_1
30          Pin_1_Write(~Pin_1_Read());    // write the value to Pin_1
31          CyDelay(250);                  // delay in milliseconds
32          Pin_1_Write(~Pin_1_Read());    // write the value to Pin_1
33          CyDelay(250);                  // delay in milliseconds
34          Pin_1_Write(~Pin_1_Read());    // write the value to Pin_1
35          CyDelay(250);                  // delay in
36          Pin_1_Write(~Pin_1_Read());    // write the value to Pin_1
37          CyDelay(750);                  // delay in milliseconds
38          Pin_1_Write(~Pin_1_Read());    // write the value to Pin_1
39          CyDelay(750);                  // delay in milliseconds
40          Pin_1_Write(~Pin_1_Read());    // write the value to Pin_1
41          CyDelay(750);                  // delay in milliseconds
42          Pin_1_Write(~Pin_1_Read());    // write the value to Pin_1
43          CyDelay(750);                  // delay in milliseconds
44          Pin_1_Write(~Pin_1_Read());    // write the value to Pin_1
45          CyDelay(750);                  // delay in milliseconds
46          Pin_1_Write(~Pin_1_Read());    // write the value to Pin_1
47          CyDelay(750);                  // delay in milliseconds
48          Pin_1_Write(~Pin_1_Read());    // write the value to Pin_1
49          CyDelay(250);                  // delay in milliseconds
50          Pin_1_Write(~Pin_1_Read());    // write the value to Pin_1
51          CyDelay(250);                  // delay in milliseconds
52          Pin_1_Write(~Pin_1_Read());    // write the value to Pin_1
53          CyDelay(250);                  // delay in milliseconds
54          Pin_1_Write(~Pin_1_Read());    // write the value to Pin_1
55          CyDelay(250);                  // delay in milliseconds
56          Pin_1_Write(~Pin_1_Read());    // write the value to Pin_1
57          CyDelay(250);                  // delay in milliseconds
58          Pin_1_Write(~Pin_1_Read());    // write the value to Pin_1
59          CyDelay(250);                  // delay in milliseconds
60          CyDelay(1000);
61      }
62  }
63
64  /* [] END OF FILE */
```