TITLE Intro, Exercise 1 PROJECT 6.115 Lab 2

~~Continued from page~~

Goals: - Modify MINMON to have read/write capabilities
- Reverse Assemble a program
- Strike a light (fluorescent lamp) manually with a signal generator, and automatically with the R31JP finding and generating the resonant frequency, with a countdown, 8254 and feedback circuit
- Understand 8051 hardware and the PSoC

Exercises (1) Add read/write to MINMON
(2) Reverse assemble a hex file
(3) Use read/write capabilities to read 2 sequential locations and write the sum, difference, product + quotient into sequential locations
(4) Turn on + study lamp with power supply and signal generator
(5) Use R31JP to create square waves - single byte countdown, double byte countdown and internal timer interrupt. Analyze timing and frequency
(6) Set up 8254 counter/timer to provide precise frequencies
(7) Create feedback circuit that gives rising + falling edge which can be used as signal for software to ~~turn on/off maintain~~ find + maintain resonant frequency
(8) Study 8051 in another single board computer setup
(9) Use and understand the PSoC with a blinky LED program

Exercise 1: All code is in Appendix A:
The main things that I changed were redirect r + w in the jump table, slightly modified crlf so it preserves the accumulator value and ~~DCFL~~ create read/write functionality by storing relevant locations in the data pointer.
We can not use w to write to any locations in ROM which makes sense because ROM is readonly.

Continued to page 10

SIGNATURE [signature] DATE 2/16/15

DISCLOSED TO AND UNDERSTOOD BY DATE

Lilly Chin

BOOK - 10 PAGE

TITLE Exercise 1, Exercise 2 + 3 PROJECT 6.115 Lab 2

Continued from page 9

Further comments

- Using the LST of the code and MINMON was helpful to test read functionality
- To release the ROM, press the silver switch, NOT the red capacitor
- calf was overwriting the value in the accumulator which is why I remote it to be safer

Exercise 2 Reassembled code in Appendix B

This program ~~does~~ turns on LED #2 on port 0. The key trick is that the program uses 0E0h instead of the shorthand for accumulator as mov A, X and mov X, a have special optimized functions with different opcodes.

Comments

- p. 2-25 has a list of commands sorted by hex number (which I found too late)
- The tip off to figuring out the last writing bits was that 20 was placed with enough frequency to look like spaces
- p. 2-9 has the initial values of all the special function registers at startup.

Exercise 3: All code in Appendix C

I added a cute loop for the user to interact with, including setting the location to start reading from. Most of the Read Num, write num code is taken from the read write functionality in Ex. 1.

The key part of my code uses data pointer to track the desired read/write position. Thus, I had to ensure that its value was set after using Print (because that function uses ~~and/or~~ the datapointer). I also forgot the termination character for .db which led to a lot of junk.

Continued to page 11

SIGNATURE

DATE 2/16/15

DISCLOSED TO AND UNDERSTOOD BY

DATE

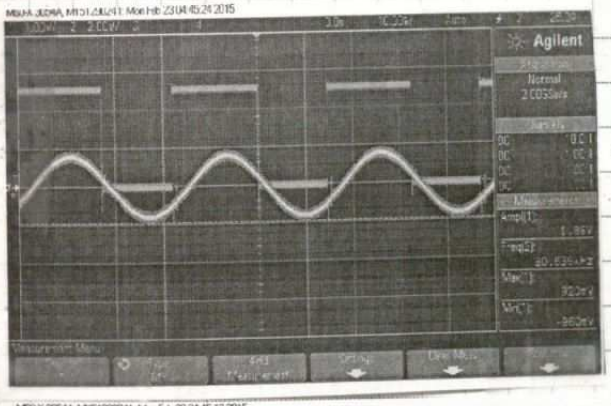
PROPRIETARY INFORMATION

Continued from page 10

Exercise 4: Connection to the lamp went as follows

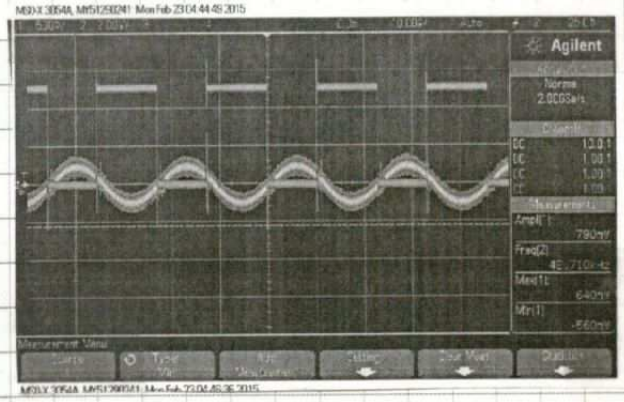
1) Plug in DC adapter for lamp box to wall
2) Tie Grounds of scope, kit, and power supply to lamp's GND
3) Verify kit's square wave generator to go from 0-5V and connect to "Square Wave"
4) Connect lab supply to line marked "40 V"
5) Set Oscilloscope to Vout and measure when on

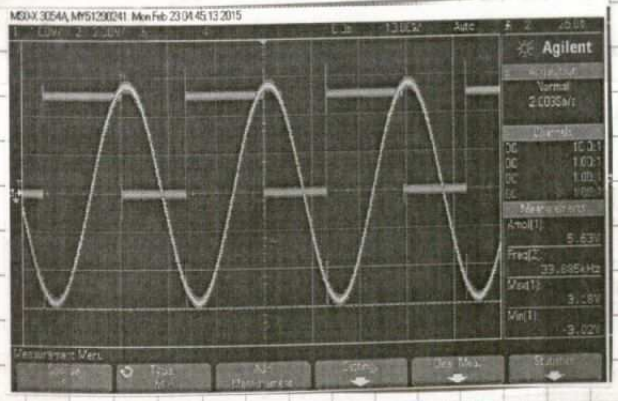
After turning the ~~Square wave sig~~ power supply to 3V, we found that the resonant frequency was around 34 kHz



30 kHz ←

→ 43 kHz





← Resonant frequency graph.

Note that the voltage waveform is sinusoidal. This is because the LC circuit smooths out the sharp transitions of the input square wave. The amplitude of the voltage peaks at the resonant frequency at 5.6V. The resonant frequency is given by $\frac{1}{\sqrt{LC}}$ which can be found by using diff eqs to solve the LC circuit.

When we take the lab supply to 40V & suddenly turn it on/off again, we first see the wave form briefly have a very high amplitude before settling down to a lower voltage. This is consistent with the light being on glow discharge before switching to a clean, bright ignition.

Continued to page 12

SIGNATURE

DATE 2/24/15

DISCLOSED TO AND UNDERSTOOD BY

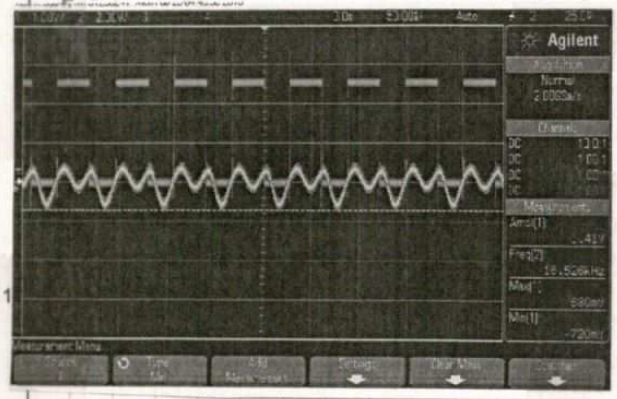
DATE

PROPRIETARY INFORMATION

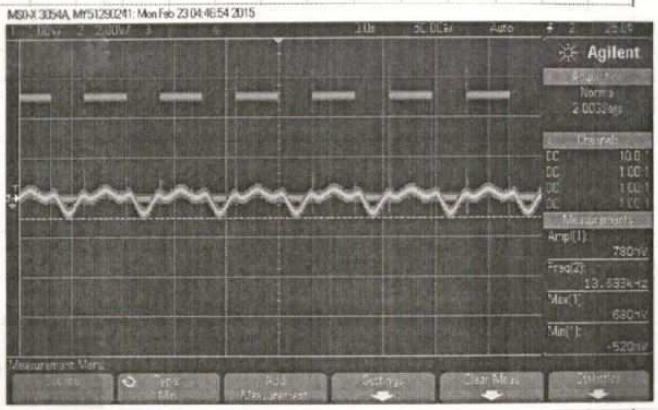
TITLE Exercise 4 + 5 PROJECT 6.115 Lab # 2

Continued from page 11

When the frequency goes below 20 Hz, we see really strange periodic signals. (Images taken with $V_{cc} = 3$ V)



~~16.5 kHz~~ 13.5 kHz →



This strange behavior comes from the fact that this is lower than the refresh rate of the LC circuit so some elements are not getting charged all the way and the underlying square wave affects the signal more.

My main ~~some~~ complications was not being able to tell the difference between glow discharge and a clean ignition, causing me to burn out a lamp (magic smoke smell) and annoying Steve Leeb. I also used a power supply that couldn't hit 40 V. I also had a couple of issues with probe tuning and automatic measurements being inaccurate

Exercise 5: All code is in Appendix D

	1 Byte	2 Byte	Interrupt Timer
~~Lowest~~ Freq.	893.0 Hz (00)	3.49 Hz (0000)	1.8 kHz (00)
~~Highest~~ Freq.	76.8 kHz (FF)	28.8 kHz (0000)	65.8 kHz (FF)
Resolution	Not good → large jumps at high frequencies (>20kHz)	Good → gradations of 3 kHz	Bad for high, but generally good → gradations of 10 Hz
Turn on loop	~~Number No:~~ 04 → 38.4 kHz; 05 → 32.9 kHz (Not enough resolution)	No, not high enough	~~Yes~~ No: F2 → 32.9 kHz; F3 → 35.45 kHz (Not enough resolution)

Overall, the 1 byte + 2 byte work by using a djnz loop for the number stored in memory, revealing why 00 acts like 256. (Decrease first, then check). The timer interrupt instead uses the number in memory as the ~~time~~ (approximate) count length of half a period

Continued to page 13

SIGNATURE [signature] DATE 2/24/15

DISCLOSED TO AND UNDERSTOOD BY DATE

Lilly Chin

TITLE Exercise 5 & 6 **PROJECT** 6.115 Lab 2

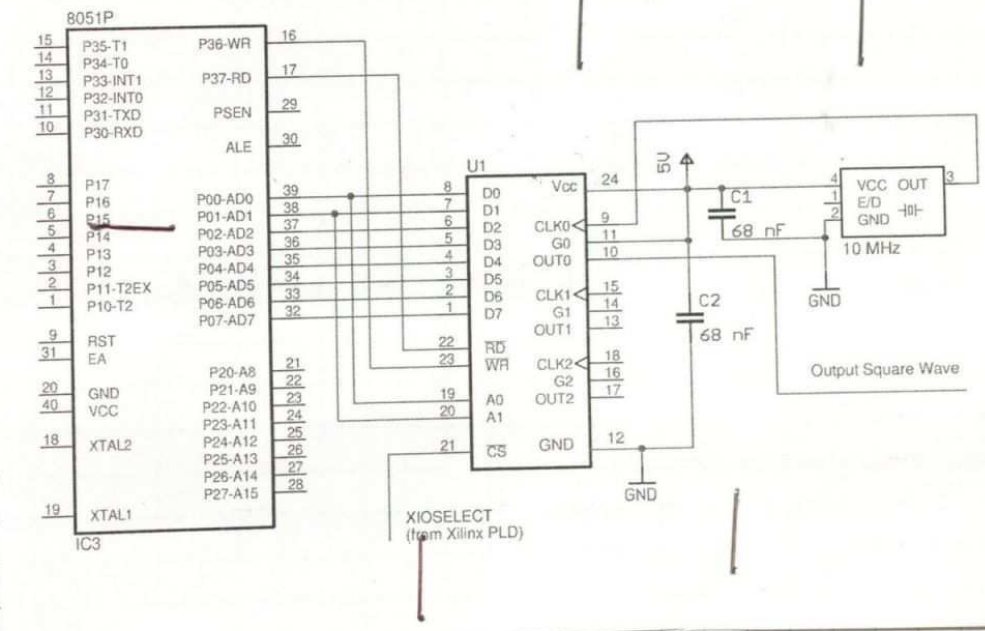
Continued from page 12

Timing analyses for the code is broken down on Appendix D

One issue I encountered with interrupt code was not including a ljmp to my code, so the ISR was being run regardless.

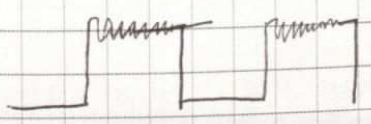
Exercise 6:

Note: I am not 100% sure where D7-D0 and A16-A0 correspond to on the R31JP so I linked them to the 8051.



It took me a while to understand that the 8254 were wired to the same things on the R31JP so when a write occured on the R31JP, the 8254 would use D0-D7 as the codeword/data for itself while using the least significant bits of the address to pick which register to write to.

I included a 68nF bypass capacitor on the circuit, I was also instructed to avoid using alligator clips because they were designed for output, not input and were thus subject to more noise. Regardless, the signal produced was not ~~enough~~ entirely square. The result looked like



but was still sufficiently square to strike the lamp. I also added a bypass capacitor of 68nF to attempt to reduce the noise

Continued to page 14

SIGNATURE

DATE 2/24/15

DISCLOSED TO AND UNDERSTOOD BY

DATE

PROPRIETARY INFORMATION

Appendix E: 6.115 Lab 2, Exercise 7, Automatic Lamp Striker

This could be run in either G8000 or MON/RUN

Automatic Lamp Striker

```
; Auto lamp striker
; This program starts at a high frequency - 50 kHz
; Then, it steps down until it sees a LOW Schmitt output
; Once it sees that low output, it steps down in frequency more slowly with finer resolution
; Once it sees a HIGH value, then it stops and maintains that frequency

; The code assumes that the Schmitt inverter output is in P1.0
; The code also assumes that the 8254 is wired up and uses writeNum to control it

; The lab doesn't want us to use interrupts (it wants us to wire the schmitt output to P1
; and not P3), so we are not using interrupts

start:
    lcall initConfig
coarseLoop:
    mov a, #10h             ; Choose value to coarsely add by
    lcall addStack
    lcall writeTimer0       ; Add the incremented value to Timer 0
    lcall wait              ; Slow down so that have enough time for circuit to catch
    jb P1.0, coarseLoop     ; If Schmitt output is not LOW, then continue coarse search
fineLoop:
    mov a, #01h             ; Choose value to finely add by
    lcall addStack
    lcall writeTimer0       ; Add the incremented value to Timer 0
    lcall wait
    jnb P1.0, fineLoop      ; If Schmitt output is not HIGH, then continue fine search
struckLoop:
    sjmp struckLoop

;*****************************************************************
; Subroutines
;*****************************************************************
;================================================================
; subroutine: initConfig
; this routine does the initial bookkeeping like insuring P1 is ready to read
; It also handles the initial start up of the 8254 with all of the code words
```

```
;=============================================================
initConfig:
    pop 06h
    pop 07h
    mov P1, #0FFh       ; Set P1 to high so we don't fry Port 1
    setb P3.5           ; Set "Output Enable" of 74C922 to 1,
                        ; so won't try to simultaneously drive P1
    mov dph, #0FEh      ; configure the 8254 to be on Mode 3 (Square Wave)
                        ; and taking the LSB and MSB for writing values
    mov dpl, #003h
    mov a, #36h
    lcall writeNum
    mov a, #0A0h        ; Write the value #00A0 into Timer 0 of the 8254
    push acc            ; This sets it to initially generate a square wave of 62.5 kHz
    mov a, #00h
    push acc
    lcall writeTimer0
    push 07h
    push 06h
    ret

;=============================================================
; subroutine: writeNum
; this routine writes the value of the accumulator into the area specified by DPTR
; It is used to configure the 8254 because it is listening to the relevant ports
;=============================================================
writeNum:
   movx  @dptr, a         ; replace value in byte with a
   ret

;=============================================================
; subroutine: writeTimer0
; this routine writes a 16 bit value from the stack to Timer 0
; Since we are changing the frequency so much,
; I decided I might as well make this a dedicated function
; Note that this uses a lot of registers
;=============================================================
writeTimer0:
    pop 00h             ; Save return address
    pop 01h
    pop 02h             ; R2 -> high byte
    pop 03h             ; R3 -> low byte
```

```
    mov dph, #0FEh      ; Address Timer 0
    mov dpl, #00h
    mov a, r3
    lcall writeNum      ; Write low byte
    mov a, r2
    lcall writeNum      ; Write high byte
    push 03h
    push 02h
    push 01h            ; Restore return address
    push 00h
    ret


;===============================================================
; subroutine: addStack
; this routine adds the first value on the stack to the 16 bit number
; specified by the 2 bytes below it on the stack.
; The routine should also handle overflows
;===============================================================
addStack:
    pop 00h             ; Save return address
    pop 01h
    pop 02h             ; R2 -> high byte
    pop 03h             ; R3 -> low byte
    clr c
    add a, r3           ; Add increment to low byte
    mov r3, a
    clr a
    addc a, r2          ; If there was a carry, add that to high byte
    mov r2, a
    push 03h
    push 02h
    push 01h            ; Restore return address
    push 00h
    ret

;===============================================================
; subroutine: wait
; this routine uses a nested loop in order to spend a few cycles
; Using this routine, we give enough time for the 8254 to output
; the new desired frequency
;===============================================================
```

```
wait:
    mov r6, #0h
    mov r7, #20h
wait_1:
    djnz r6, wait_1
    djnz r7, wait_1
    ret
```