Example logbook report for Apsc 1299:

The Morse Decoder

This following pages contain an example logbook that a student might have written while performing an experiment, called the “Morse Decoder”, where they created code to translate Morse code digits into regular Arabic digits. This example logbook is particularly useful for seeing how to record your work while coding and while troubleshooting code that is not behaving correctly.

Note that you do *not* need to write your logbook exactly like this one in order to get full marks—there is quite a bit of flexibility in how an engineer can organize and record data in their logbook. However, this example should give you an idea of what level of detail you need to aim for, and what kinds of data are important to record.

An engineer’s logbook can be used as a legal document to, for example, establish who owns an idea or innovation in the case of a patent dispute. For this reason, your logbook must be written in ink, and have each page signed (or initialed) and dated. Graphs may be done in pencil. Computer code should be permanently pasted or stapled into the book.

Your logbook also needs to be a complete record of everything you did in the lab, so even your mistakes and unwanted results should be documented fully. Never remove pages, and never obscure information from the logbook. If you make a mistake, cross out the error with a single line (so it can still be read), and write the correction nearby.

When you join the work force, you’ll often be called upon to make an estimate on how long a job will take, or to bill your clients accurately for the time you spent on their project. For this reason, you’re required to record the current time in your logbook regularly. By getting into this habit now, you’ll develop a good “feel” for how long certain tasks take, and will always have a record of exactly how long you worked on a particular project.

The annotations in this logbook will point out things an instructor would think the student did well and things they would feel were lacking. Overall, however, this logbook displays good note-taking skills and would likely receive a very good grade.

Ima-Jean Yuss
Dec 4, 2013

Sign and date every page

MORSE DECODER

Equipment	Files
PIC 18F 4525 MCU	morseDecoder.txt
breadboard, wires	SerLCD.c, SerLCD.h
switch	configuration_bits.h
LCD screen	OSC.c, OSC.h
10 KΩ resistor	configureUSART.c
200 Ω resistor	configureUSART.h

Start a new experiment's write-up by recording the experiment title and all the equipment and computer programs you'll use.

Starting to read manual 9:38 AM

Record the time often.

Notes:

- want to create code that will decode button pushes in Morse code into regular Arabic digit.
    - will only do it for Morse code numbers because they always have five dots or dashes. The alphabet's codes can be longer or shorter, so it's harder to write code for
- Morse code numbers are:

1 = 01111	6 = 10000
2 = 00111	7 = 11000
3 = 00011	8 = 11100
4 = 00001	9 = 11110
5 = 00000	0 = 11111

9:43 AM

The student takes notes as she reads through the manual's theory, recording all the important information in her logbook.

Tasks:

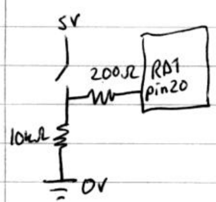
- use morseDecoder.txt to create needed code
- will need to add a lot to make it work
- use timers to test whether a button push is long enough to be considered a dash
    - use 0.35 s or longer as a dash
    - will include 1/10th sec delay after button is pushed or released to prevent "button bounce" errors
- code must accept button push of up to 2 sec with no errors. 9:47 AM

When she gets to the tasks, she jots them down in point form. This is a clear way to state her experimental objectives.

If you don't record the time at least once per page, you'll lose marks.

Ina-Jean Yuss
Dec 4, 2013

- Button is already set up from Lab 6, Timers 9:48AM
  - LCD still attached too



- Creating project using procedure on last page of logbook 9:50Am
  include files: serLCD.c, serLCD.h
  configuration_bits.h

Consider putting the steps for often-performed tasks in an appendix of your logbook. That way, you can refer to it rather than re-writing the steps.

- Creating a file called morseDecoder.c from supplied file morseDecoder.txt 9:54AM

- Going to need to add code in switch1_risingedge_action() function to time how long button push is
  - I'll add a timer that runs for as long as button is pushed. Then, ReadTimer0() and see how long button was pushed

⇒ In Lab 6, Timers, I found a prescaler of 256 allows me to time intervals up to $256 \times T_{cy} \times 2^{16} = 2.097$ sec 9:59Am

Record your plans, rough work, and thought processes as you prepare for a task.

- Want a total time of 0.35 sec for minimum button push length to count as a dash, not a dot.
- Code has a delay of 0.1sec to prevent "button bounce" errors, so timer has to run for 0.25s at least for a dash.

$$0.25\text{ sec} = Y \times \underset{\text{prescaler}}{256} \times \underset{T_{cy}}{125\text{ns}}$$

$$\Rightarrow Y = \frac{0.25}{256 \cdot 125 \times 10^{-9}} = 7812.5 \approx 7813$$

↳ So I'll need to test wheter ReadTimer0() got to 7813 or higher

10:06Am

Irva-Jean Yuas
Dec 4, 2013

- Copy/pasted the OpenTimer command from Lab6 using-timers.c code 10:06AM
- Changed prescalar to 256
- I only need to open timer once, and never close it, so I put OpenTimer in main() but outside of the while(1) loop 10:07AM

- Now need to build code in switch1-risingedge-action()
  - Add WriteTimerØ(Ø);
  - Added a while (PORTDbits.RD1) loop to just wait for the button to stop being pressed

- Next will read timer and test whether it got past 7813. If yes, record 1. If no, record Ø
  - have to record values in an array: dotdash-array[ ]
  - They defined a global variable i in the code to act as the index for dotdash-array[i]

So this is what I'll do:

```
if (ReadTimerØ() > 7813)
{ dotdash-array[i] = 1;
  i++; }
else
{ dotdash-array[i] = 0;
  i++; }
```
10:14AM

- Resetting i to zero is taken care of in the supplied code (i.e. 5 button pushes is one digit in Morse)

- Now adding this stuff to switch1-risingedge-action() function.

10:16AM

Think of the logbook as a real-time diary of your work. Record what you do, as you're doing it.

Notice how the student jots down notes on everything she does, step by step.

Here, the student jots down some pseudo-code as part of her planning process for the project.

Ima-Jean Yuss
Dec 4, 2013

-Now adding comments to code 10:18AM

-Now creating code to display this text on LCD:
Morse: <5digit code>
Arabic: <one digit> "

-Added LCD-Clear() to clear screen
-Added #include "..\Functions\serLCD.h"
-Added LCD_SetPosition (1,1);

```
printf("Morse: %i%i%i%i%i", dotdash_array[0],
                             dotdash_array[1],
                             :
                             dotdash_array[4]);
```

10:24AM

-Now adding code to display the Arabic digit 10:27AM
-Added LCD-SetPosition(2,1);

```
printf("Arabic : %i", ?? )
```

need a variable here that is the correct number

-Okay, so I first need code to translate from Morse into regular numbers
- I'll take the array numbers and turn them into a binary number. Then I can use its value to trigger a case switch.
- I think I'll put that in a separate function called getArabic (morse) where morse will be the binary number I got from the array

10:32AM

-So:

```
morse = dotdash_array[0]x1 +
        dotdash_array[1]x2 +
        "        "   [2]x4 +
        "        "   [3]x8 +
        "        "   [4]x16
```

10:33AM

Again, this is a step-by-step, real-time diary of what she's doing, as she does it.

Someone could take the same code she started with and reproduce her work using only the notes in this logbook, with no lab manual. This is the level of detail you should be aiming for.

Ima-Jean Yuss
Dec 4, 2013

- Added ~~morse~~ calculation of "morse", as on previous page, to print_morse() function 10:34AM
- declare morse as type char

- Creating getArabic(morse) function 10:38AM

using XXXXX, I have:
↓ ↓ ↓ ↓ ↓
x16 x8 x4 x2 x1

1 = 01111 = 15 = 8+4+2+1
2 = 00111 = 7 = 4+2+1
3 = 00011 = 3 = 2+1
4 = 00001 = 1 = 1
5 = 00000 = 0 = 0
6 = 10000 = 16 = 16
7 = 11000 = 24 = 16+8
8 = 11100 = 28 = 16+8+4
9 = 11110 = 30 = 16+8+4+2
0 = 11111 = 31 = 16+8+4+2+1

- Created a case switch in getArabic() to return correct digit
  ↳ What to return if none of these?
  ↳ okay, I made the return value of type unsigned int, so I can return a larger value than ∅-9. I'll use 20 as the error code / default value for switch.
  - I'll also need to add some new variable in print_morse() to catch a "20" and display an error message 10:47AM

- Added "unsigned int testForError" declaration to print_morse() function 10:49AM

- Added if/else loop to print "error" if 20 is returned, and print the value otherwise
10:53AM

More diary of her work, interrupted twice when she pauses to figure out something.

Irna-Jean Yuas
Dec 4, 2013

- Now creating case switch in getArabic() 10:53AM
- Done creating switch, ready to build ~~10~~ 11:00AM

PKWARN 0003: Unexpected device ID
↳ forgot to turn on power to breadboard

Build failed: Warning 2066 – type qualifier mismatch
" " x 12 occurences
↳ They're all on printf commands. I forgot to change the memory model to "large code"
Project > Build options > Project > MPLAB C18 > Categories: memory model > Select: large code model
11:04AM

Build failed: could not find definition of main
↳ Oh. I created morseDecoder.c but forgot to put it into the project! 11:05AM

Build failed: syntax error
↳ forgot a semi-colon on line 110 11:06AM

Build failed: syntax error
↳ forgot semi-colon on line 137 11:07AM

Build failed: ① Error 1105 – symbol 'TIMER_INT_OFF' not defined
" " x 4 occurences
② Warning 2058 – call of function without prototype
" " x 10 occurences
③ Error 1109 – type mismatch in getArabic()
11:09AM

① forgot #include <timers.h>
② forgot #include <delays.h>
11:10AM

A common error is to not record enough notes when things start to go wrong.

Notice how this student handles coding problems: She notes every error message, including its code and warning.

Then, as she figures out what went wrong, she notes what the problem was and how to fix it.

It's implied, here, that she made the fixes she noted. This is okay for simple problems. If a more complex solution was required, she would have needed to add more details about what she did.

Ima-Jean Yuas
Dec 4, 2013

Build failed ① Warning 2058 - Call of function without prototype 11:11 AM
" " x 5 occurences

② Error 1109 - type mismatch in getArabic()

① This is on printf
↳ Ah! Need #include<stdio.h> to use printf

Build failed: ① Warning 2058 - Call of function without prototype
② Error 1109 - type mismatch in get Arabic()
11:13 AM

① On line "testForError = getArabic(morse);"
↳ I forgot to put in a prototype for the new function getArabic()

② I had testForError as type char when getArabic() returns an unsigned int to testForError

Build succeeded → no errors or warnings 11:17 AM
Program target device

- Program not working. Nothing shows up on screen

- I know the LCD is working so I'll use in-circuit debugging to check whether code is working
Debugger > Select tool > Pickit 2
↳ say yes or OK to all the dialogue pop-up boxes

Put breakpoints inside print_morse()
switch1_risingedge_action()
getArabic()

11:21 AM

Here she encounters a different kind of problem: Her project built successfully, but the circuit is not behaving as intended.

She jots down notes on everything she does to fix the problem, step by step. This is the same level of detail she used when she documented creating her code.

Ima-Jean Yuss
Dec 4, 2013

- Stepping through code... 11:24 AM
  - program goes into switch1_risingedge_action() five times for five button presses, which is correct
  - Then it goes to get Arabic(), which is correct

Observation! → my LCD screen is full of gibberish

  - Then program goes back to recording digits because it goes back into switch1_risingedge_action()

The gibberish on screen is because I didn't
→ configure USART! Argh, argh, argh... 11:34 AM

- Added configureUSART.c and configureUSART.h to project
- Added #include "..\Functions\configureUSART.h"
- Added lines:

```
set_osc_32MHz();
configureUSART(9600ul, 32);
```

Build failed: couldn't find definition of set_osc_32MHz
11:39 AM

?? That's in osc.h, and the provided file has the # include statement for it. Don't know why this is happening...
↳ Ack! It's not in the project! Forgot to add it
  - adding osc.h, osc.c

Build succeeded, no errors
& Program target device 11:42 AM

Yah! Five taps of the button gives a correct-looking display. It gives " Morse: 11111
Arabic: Ø "
11:43 AM

When you're testing or measuring something, record all your observations fully.

It's not enough to say something worked or didn't work. Record what you observed that led you to that conclusion.

Irma-Jean Yuas
Dec 4, 2013

Testing to make sure all numbers translate correctly: 11:43AM

Tap pattern	Morse:	Arabic:	
1 = 01111	01111	9	incorrect!
2 = 00111	00111	8	
3 = 00011	00011	7	
4 = 00001	00001	6	
5 = 00000	00000	5	✓ correct
6 = 10000	10000	4	incorrect!
7 = 11000	11000	3	
8 = 11100	11100	2	
9 = 11110	11110	1	
0 = 11111	11111	0	✓ correct
10101	10101	error!	✓correct
01000	01000	error!	
00010	00010	error!	

Again, when testing or measuring something, record all your observations.

- I have to be careful with fast taps. It's very touchy and I still get "button bounce" errors a lot.

- So Morse numbers are fine. Only Arabic numbers aren't being translated correctly. The problem is probably in my case switch or my math. 11:56AM

- Okay, yeah. I display Morse code as $\overbrace{[0][1][2][3][4]}^{\text{dotdash\_array[ ]}}$
So I need to change how I multiply these elements to convert to a number. Correct is:
[0]×16 + [1]×8 + [2]×4 + [3]×2 + [4]×1

- Changed ~~definit~~ assignment to morse variable to do this

11:59AM

Ima-Jean Yass
Dec 4, 2013

Build succeeded > Program target device 12:00 PM

- New results:

Tap pattern	Morse:	Arabic:
1 = 01111	01111	1
2 = 00111	00111	2
3 = 00011	00011	3
4 = 00001	00001	4
5 = 00000	00000	5
6 = 10000	10000	6
7 = 11000	11000	7
8 = 11100	11100	8
9 = 11110	11110	9
0 = 11111	11111	0

✓ correct!

Note: When it triggers correctly, it's fine, but I get a lot of errors due to button. A long press at the start seems more likely to generate an error.

- Otherwise, works as intended!
  - Displays: "Morse: (correct Morse code)
    Arabic: (correct digit or "error!")"

  in response to Morse code numbers being tapped in with button pushes. 12:06 AM

Again, record the observations that led you to believe the circuit or code is working (or not working) as intended.

- Code ~~are~~ attached on next page
  ↳ Wait! Adding more comments and re-printing 12:07 PM

Always attach your code to your logbook. The level of detail you should aim for is that someone could reproduce the experiment based only on your notes. They would need a copy of your code to do that.

Before you print out, however, double-check whether you included enough commenting. Comments are one of the easiest things to forget!

```c
////////////////////////////////////////////////////////////////////
// Barebones code to display Morse code as Arabic numbers on LCD
//
// created Dec 2013 by Jen DeBenedictis
//
// Morse code numbers:
// (where 0s = dots, 1s = dashes)
//  01111 = 1
//  00111 = 2
//  00011 = 3
//  00001 = 4
//  00000 = 5
//  10000 = 6
//  11000 = 7
//  11100 = 8
//  11110 = 9
//  11111 = 0
//
// lots of changes required to make this code work!
//
////////////////////////////////////////////////////////////////////

#include <p18f4525.h>
#include <timers.h>
#include <delays.h>
#include <stdio.h>
#include "..\Functions\osc.h"                // library for set_osc_32MHz()
#include "..\Functions\configuration_bits.h" // configures bits on MCU
#include "..\Functions\serLCD.h"            // for LCD commands
#include "..\Functions\configureUSART.h"    // for configuring LCD

void monitor_switch1_for_edges(unsigned char digitalinputpin);
void switch1_risingedge_action(unsigned char digitalinputpin);
void switch1_fallingedge_action(unsigned char digitalinputpin);
unsigned int getArabic(char morse);
void print_morse(void);

unsigned char last_switch1_edge = 0;          // last edge, start with low
unsigned char switch1_risingedge_found = 0;
unsigned char switch1_fallingedge_found = 0;

unsigned char dotdash_array[]={1,1,1,1,1};  // Array to hold the Morse code numbers
unsigned char i=0;                          // i will be used as the array index

void main(void)
{
    set_osc_32MHz();            // select 32 MHz oscillator speed
    configureUSART(9600ul,32);  // configure LCD to communicate with MCU

    TRISDbits.TRISD1 = 1;       // Configure pin 20, RD1, as a button

    OpenTimer0(TIMER_INT_OFF & T0_SOURCE_INT & T0_16BIT & T0_PS_1_32);

    while(1)
    {
        if (i==5)   // a full Morse code number has been received after 5 button pushes
        {
            print_morse();  // function to print the number on the LCD
            i=0;            // reset the array index
        }

        monitor_switch1_for_edges(PORTDbits.RD1); // detect button push or release
    }
}


// general button function
void monitor_switch1_for_edges(unsigned char digitalinputpin)
{
    if (last_switch1_edge == 0 && digitalinputpin )
    {   // rising edge detected if digitalinputpin is 1 (on)
```

Code should have enough commenting that someone could understand how the program works based only on the comments.

In your opinion, does this code have enough comments?

```
            switch1_risingedge_action(digitalinputpin);
            last_switch1_edge = 1;  // rising switch edge detected
            switch1_risingedge_found = 1;
        }
        else
        {
            switch1_risingedge_found = 0;
        }

        if (last_switch1_edge == 1 && !digitalinputpin )
        {   // falling edge detected if digitalinputpin is 0 (off)
            switch1_fallingedge_action(digitalinputpin);
            last_switch1_edge = 0;  // falling switch edge detected
            switch1_fallingedge_found = 1;
        }
        else
        {
            switch1_fallingedge_found = 0;
        }
}

void switch1_risingedge_action(unsigned char digitalinputpin)
{
    Delay10KTCYx(80);   // pause for 0.1 s to reduce "button bounce" errors

    // Insert code to time how long the button push is.
    // Taking into account the 0.1 s delay above, make any
    // button push that is longer than 0.35 s register as a
    // "dash" and any button push shorter than that as a "dot".
    // Ensure that button presses up to 2s in length won't
    // cause errors due to memory limitations.

    // Next, write your data to the array dotdash_array[i].
    // Dashes should be recorded as 1s and dots should be
    // recorded as 0s.

    WriteTimer0(0);
    while(PORTDbits.RD1){}; // Wait for button to stop being pressed

    if(ReadTimer0() > 7813) // (7813-0)*256*125ns = 0.25 s.
    {                       // 0.25s + 0.1s delay above = 0.35s = dash, not dot
        dotdash_array[i]=1;
        i++;
    }
    else
    {
        dotdash_array[i]=0; // shorter than 0.35s = dot, not dash
        i++;
    }
}

void switch1_fallingedge_action(unsigned char digitalinputpin)
{
    Delay10KTCYx(80);   // delay 0.1 s to avoid "button bounce" errors
    // do nothing else
}

void print_morse(void)
{
    char morse;
    unsigned int testForError;

    // Insert code to display the Morse code number on the first
    // line of the LCD with dots displayed as 0s and dashes
    // displayed as 1s. Put the heading "Morse: " before the number.

    LCD_Clear();                // clear LCD
    LCD_SetPosition(1,1);       // set to 1st line, 1st space of LCD
    printf("Morse: %i%i%i%i", dotdash_array[0],    // print Morse code
                              dotdash_array[1],
                              dotdash_array[2],
```

These were the comments in the original, supplied code.

The student probably should have changed them, but she included her own comments after her added code, so all the necessary information is here.

```
                                   dotdash_array[3],
                                   dotdash_array[4]);

    // Insert code to display the corresponding Arabic digit on the
    // second line of the LCD. Put the heading "Arabic: " before the digit.

    morse = dotdash_array[0]*16+    // convert Morse code into a number
            dotdash_array[1]*8+
            dotdash_array[2]*4+
            dotdash_array[3]*2+
            dotdash_array[4]*1;

    testForError = getArabic(morse);    // converts Morse to Arabic

    LCD_SetPosition(2,1);   // set to 2nd line, 1st space of LCD
    printf("Arabic: ");

    if(testForError==20)
    {
        printf("error!");   // print error message if not Morse code
    }
    else
    {
        printf("%i",testForError); // print Arabic digit
    }
}

unsigned int getArabic(char morse)
{   // function translates Morse code number into Arabic number
    // and returns it

    switch(morse)   // Translates the Morse code and returns
    {               // an Arabic digit or an error code (20)
        case 15:
            return 1;
            break;
        case 7:
            return 2;
            break;
        case 3:
            return 3;
            break;
        case 1:
            return 4;
            break;
        case 0:
            return 5;
            break;
        case 16:
            return 6;
            break;
        case 24:
            return 7;
            break;
        case 28:
            return 8;
            break;
        case 30:
            return 9;
            break;
        case 31:
            return 0;
            break;
        default:
            return 20;
            break;
    }
}
```