EE 109
Debugging Techniques

Debugging Techniques

It Doesn’t Work. Now What?

• Debugging an embedded system can be difficult.
  • Not intended for users to see how things are done.
  • Can be hard to check value of variables.
  • Often can’t single-step the program and examine registers or memory contents.
• Big question: Is it a software or a hardware problem, or both?
• Advice: Don’t rely on just fiddling with the code.
  • Make use of the test equipment

Debugging Techniques

Where do we start?

• First, check the obvious
  • Is there power to the circuit?
  • Is there a clock signal to the µC?
  • For Arduino and other development boards, if you were able to download the program then power and clock are present.
  • For other designs, use test equipment like the multimeters and oscilloscopes to check power supplies and clocks.

The µC is running my code but nothing is happening

• The key to debugging is to isolate problems.
• Bouncing around the program making changes is ineffective since you don’t know what works and what doesn’t.
• Try to find some part of the program that is working.
  • Comment out routines doing more complicated tasks and see if the simple stuff is functioning.
• When in doubt, revert to a version that worked.
Debugging Techniques

Make things easier on yourself.

- When possible don’t combine code that affects multiple components in a single routine.
  - Example: A program reads from an ADC and writes to a serial port.
  - Have one function for reading from the ADC
  - A second function for writing to the serial port.
- Write test programs that exercise specific parts of your design, and don’t touch the other parts.
  - When a component of your full design isn’t working, try the test program to confirm that it works by itself.

Debugging Techniques

Your best friend: test equipment

- Changing the software may not tell you much.
  - If you have no idea why the program isn’t working, making changes to the code can be a waste of time.
- Make use of the test equipment
  - A quick look with a scope may tell you more than making numerous revisions to the code hoping to stumble onto the problem.

Debugging Techniques

Add test routines to your program while developing it.

- Use conditional compiling directives to include test code in the program when debugging.

```c
#define TESTSEG 1

int main(void)
{
  DDRD |= 0x7F;         // Set PORTD bits 0-6 for output
  #ifdef TESTSEG
  PORTD = 0x00;        // Light up all the segments
  while (1) {}
  #endif
}
```

Debugging Techniques

Use I/O bits as indicators

- How can we tell if parts of the program are executing?
- If you expect the program to be in a loop of some sort, pick an unused I/O bit and make it go 0-1-0 each time through the loop.
- Watch for it with the scope.

```c
while (z != x) {
  PORTC |= (1 << PC5);  // rest of code in loop
  PORTC &= ~(1 << PC5);
}
```
Debugging Techniques

Use I/O bits as indicators

- Use an I/O bit to show progress through the program.
  - Put the indicator at one point and use the scope to see if it pops up.
  - If so, move it somewhere else and try again.
  - Can use more than one indicator, or the same indicator multiple times.

```c
PORTC |= (1 << PC5);
PORTC &= ~(1 << PC5);
// code that may be causing problem
PORTC |= (1 << PC4);
PORTC &= ~(1 << PC4);
```

Debugging Techniques

Looking for one-time events

- The program seems to be getting stuck. How do I find out where that’s happening?
- If the problem is in a part of the program that is not in a loop, it can be harder to find.
- For example: Is some initialization code, or an interrupt service routine for some event ever being executed?
- Previous method worked with code in loop but this code is only executed once.

Debugging Techniques

Looking for one-time events

- Can look for one-time events with a scope if we use an I/O port as a signal to trigger the scope.
- Scopes can record the signal around a trigger and hold it for viewing.
- In the program
  - Add the indicator to the program at some point
  - Configure the scopes to capture a single event.
  - If the scope triggers on the event, program got to that point
  - If the scope doesn’t trigger, program never got there.

Debugging Techniques

Looking for one-time events

- May be necessary to add a one or two second delay to the start of program to give yourself time to arm the scope’s trigger.
- To acquire an event:
  1. Set the scope’s triggering level and set for a single trigger.
  2. Restart or power cycle the board.
  3. Arm the scope to wait for the trigger after the delay.
  4. When the trigger signal occurs, the scope will capture the event and hold the view.
Debugging Techniques

Case Study: The Resetting Arduino

- With interrupts disabled, the initial display on the LCD is stable.
- With interrupts enabled, the initial display is rapidly being overwritten, and nothing else works.
- Used the LED to discover that the board was restarting in a loop.

```c
main()
{
    PORTB |= (1 << PB5);
    _delay_ms(250);
    PORTB &= ~(1 << PB5);
    _delay_ms(250);
}
```

- The ADC initialization code
  ```c
  ADCSRA = 0xCB;
  ```
- Bad way to do it. This code made it difficult to see at a glance that the ADC was being enabled, interrupts enabled, and a conversion started in one statement.
- Better to do
  ```c
  ADCSRA = (1 << ADEN | 1 << ADSC | 1 << ADIE | 3 << ADPS0);
  ```

Case Study: The Resetting Arduino

- Three small errors working together
  - The ADC interrupt service routine had not yet been written
  - ADC interrupts were enabled
  - A conversion was started.
- The compiler puts a jump to the program reset in unused vectors (vectors for ISRs that are not defined.)
- Spurious interrupts cause the program to start over.
- Result: Each time the ADC finished its first conversion, it interrupts, the vector makes it jump to the reset and start over doing it again, and again, and again, etc.
- Reading assignment: Murphy’s Law