Debug Techinques for HW/SW Systems

What's broken, the HW or the SW?

Some basic thoughts:

-assume that it's broken

-it's not working correctly until proven by repeated experiments

-never quite trust that it works correctly

-assume that a few small bugs may still exist

-failure to find bugs usually stems from wrong assumptions

-when debugging, prove each assumption correct

-don't grasp at straws or take shots in the dark

-engineering is not gambling

A debugging world view:

a zeitgeist of suspicion,
tempered by trust in the laws of physics,
curiosity dulled only by the determination to stay focused on a single problem,
and a zealot's regard for the scientific method.
(*Jack Gansle*)

Debug Techinques for HW/SW Systems

What's broken, the HW or the SW?

A debugging methodology: (adoped with changes from: Gansel, The Art of Designing Embedded Systems)

- -make the bug repeatable
- -observe the behavior, getting information to isolate the apparent bug
- -create a suspect list (remain rational in this step)
- -generate a hypothesis to pinpoint the bug
- -create an experiment to test the hypothesis
- -fix the bug
- -test the fix and then everything else

You must have good ***controlability and obserability*** into the system.

- -*probe with instruments* to stimulate and observe the hardware
- -*instrument the software* to cause and observe software behavior
- -this means to both *inject stimulus* and to *observe results*
- -keep notes of your observations

Hardware instruments

- -multimeter, oscilloscope, signal generator

Software instruments

- -LEDs, “printf”, toggle a port pin, watch a timer

Debug Techinques for HW/SW Systems

More thoughts about debugging...

Sort out what the bug is. Are you looking at the problem or at a symptom?

When you find anything you cannot explain, write it down!

Is the appearance of the bug correlate with something else?

-a new piece of code, new hardware added?

-what changed since “last time”?

When generating a hypothesis:

-ask, “what could cause this behavior?”

-don't “shotgun”, use unlikely scenarios (phase of moon, color of pencil)

-no wishful thinking

A hypothesis is confirmed by observing the hypothesis. Be absolutely sure what you are seeing is what you are looking for. Double check instruments.

Once you fix the bug:

-prove the problem is gone, replace the bug, see it, then take it away again

Debug Techinques for HW/SW Systems

Thoughts worth considering...

Bugs that mysteriously go away will mysteriously reappear!

-Find the problem, or your customer will.

Random probing will often give some hints as to what is broken. Just as often you will find weird things that are completely normal and think them to be bugs.

-Avoid rabbit trails. Stick to your hypothesis.
-Make sure the power is on.
-Was the board basically functional? Run a sanity test.
-Did you compile the right source file
-"When things are acting funny, measure the amount of funny".
(Bob Pease)
-Are any parts warm or hot to the touch? Should they be?
-The best troubleshooting tool is between your ears.
-Check your grounds. even if two parts are grounded together, are they grounded to the rest? Trace all grounds to the power source with a DMM.

Debug Techinques for HW/SW Systems

What's broken, the HW or the SW?

If possible, with both new hardware and new software...

-build a little hardware,
-write a little software,
-test at each step

The sure path to frustration is to build all the hardware, write all the software and expect the whole mess to come up and work as expected.

"…reason back from the state of the crashed program to determine what could have caused this. Debugging involves backwards reasoning, like solving murder mysteries. Something impossible occurred, and the only solid information is that it really did occur. So we must think backwards from the result to discover the reasons."
Brian W. Kernighan, *The Practice of Programming*

Debug Techinques for HW/SW Systems

What's broken, the HW or the SW?

If possible, with both new hardware and new software...

-build a little hardware,
-write a little software,
-test at each step

The sure path to frustration is to build all the hardware, write all the software and expect the whole mess to come up and work as expected.

"…reason back from the state of the crashed program to determine what could have caused this. Debugging involves backwards reasoning, like solving murder mysteries. Something impossible occurred, and the only solid information is that it really did occur. So we must think backwards from the result to discover the reasons."
Brian W. Kernighan, *The Practice of Programming*

Debug Techinques for HW/SW Systems

Some more wisdom

"The single greatest asset a designer can have is self-knowledge. Knowing when your thinking feels right and when you're trying to fool yourself...Knowing your strengths and weaknesses, prowesses and prejudices...Learning when to ask questions and when to believe your answers."

Jim Williams, Linear Technology

ATMega128 I/O Ports

Building/Testing Lab 2

Build the display and then make sure the original board works. Blink some lights!

This is also known as the ***sanity test***. It helps you avoid...

- -programming problems
- -power supplies that were left off
- -obviously shorted stuff
- -101 other stupid things that are easy to do

```
#include <avr/io.h>
int main(){
   uint16_t i;
   DDRB = 0xFF;                        //port B all outputs
   while(1){
      PORTB = PORTB ^ 0xFF;            //flip all bits
      for (i=0;i<100000;i++) {}; //spin
   }
}//main
```

ATMega128 I/O Ports

Building/Testing Lab 2

Next, try testing to make sure the segments are connected correctly and that the digits are enabled correctly.

```
int main(){
DDRA  = 0xFF;   //set port A to all outputs
DDRB  = 0xF0;   //set port bits 4-7 B as outputs
DDRD  = 0x00;   //set port D all inputs
PORTD = 0xFF;   //set port D all pullups
PORTA = 0x00;   //set port A to all zeros

while(1){
  PORTB = ~PIND; //push button selects segment
  } //while
}  //main
```

ATMega128 I/O Ports

Building/Testing Lab 2

Next, try walking through the segments and digits separately.

```
uint16_t i;     //big counter
uint8_t  k, j; //small counters

while(1){
  k++;
  j = (k % 5);  //j varies from 0 to 4
  PORTB = (j << 4); //walk through the digits
  PORTA = ~0x01;  for (i=0; i<100000; i++){}  //A
  PORTA = ~0x02;  for (i=0; i<100000; i++){}  //B
  PORTA = ~0x04;  for (i=0; i<100000; i++){}
  PORTA = ~0x08;  for (i=0; i<100000; i++){}
  PORTA = ~0x10;  for (i=0; i<100000; i++){}
  PORTA = ~0x20;  for (i=0; i<100000; i++){}
  PORTA = ~0x40;  for (i=0; i<100000; i++){}  //G
  PORTA = ~0x80;  for (i=0; i<100000; i++){}  //colon
  } //while
```

ATMega128 I/O Ports

Building/Testing Lab 2

Debug scenarios (what to do?):

-One of the digits are not displaying.
-Two digits display at simultaneously.
-All the digits display continuously.
-Some segments exhibit ghosting when you bring your hand close to the board.
-The display is totally blank.
-The entire display works one time through then quits.

ATMega128 I/O Ports

Remember....