EECE 218
Microcontrollers

Practical Debugging
Key ideas

Debugging shows only the presence of bugs, not their absence.

Bugs: problems/faults, debugging: find & fix

Good design: less debugging

Errors: consider them as opportunities to learn

- about the program you are working on
- kind of mistakes you make
- quality of your code from someone else’s viewpoint
- about how you solve problems
- about how you fix errors.
Devil’s guide to debugging

- Find the error by guessing (‘trial and error’)
- Don’t waste time trying to understand the problem
- Fix the error with the most obvious fix
- Debug by superstition (HW bug! Compiler!)
  YOU are responsible for finding YOUR bugs
  (or proving that the compiler is indeed buggy)
Finding an error

General:
1. Gather data through experiments
2. Form hypothesis to account for the data
3. Design an experiment to prove/disprove hypothesis
4. Carry out the experiment
5. Repeat as needed.

Programming:
A. Stabilize the error (should be reproducible)
B. Locate the source of error (in code)
C. Fix the error
D. Test the fix
E. Look for similar errors.
Tips for finding errors

- Use all the data to make a hypothesis
- Refine test cases (make them more specific)
- Reproduce the error in several different ways
- Use the results of negative tests
- Brainstorm for hypotheses
- Narrow the suspected region of code
- Be suspicious of routines which had errors before
- Check code the changed recently
- Integrate incrementally
- Set max time for quick & dirty debugging
- Talk to someone else about the problem
- Take a break from the problem
Syntax errors

- Don’t trust the compiler’s
  - Line numbers
  - Messages
  - Second message

- Divide & conquer
  - Stub/remove unneeded parts
  - Narrow down to code fragment
Fixing an error

- Understand the problem before fixing it
- Understand the program, not just the problem
- Confirm the error diagnosis
- Relax ("Never debug standing up")
- Save the original source code
- Fix the problem, NOT the symptom
- Change the code only for good reason
- Make one change at a time
- Check your fix (with someone else, if possible)
- Look for similar problems
Psychological considerations

- People tend to see what they want to see.

\[
\begin{align*}
\text{If } (x < y) & \quad \text{If } (x < y) \\
\text{swap} = x & \quad \text{swap} = x \\
x = y & \quad x = y \\
y = \text{swap} & \quad y = \text{swap}
\end{align*}
\]

- ‘Distance’ between identifiers:

<table>
<thead>
<tr>
<th>STOPPT</th>
<th>ST0PPT</th>
<th>Bad!</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHIFTRF</td>
<td>SHIFTRT</td>
<td>Bad!</td>
</tr>
<tr>
<td>PRODUCT</td>
<td>SUM</td>
<td>Good</td>
</tr>
</tbody>
</table>

EECE 218 – Microcontrollers
Tools of the trade

- IDE-s: editor/compiler/terminal/debugger
- Compiler warnings (treat them as errors)
- Extended checks (code analyzers)
- Debuggers: - Reproducible errors
  » Check memory, breakpoints, single-stepping
  » Complex condition breaks, source code access
- For real-time / embedded products:
  » Logic analyzers (bus and other signals)
  » In-circuit emulator (emulates uC hardware)
  » BDM: master processor controlling embedded uC