Tracing Interrupts in Embedded Software

Giovani Gracioli
Federal University of Santa Catarina
giovani@lisha.ufsc.br

Sebastian Fischmeister
University of Waterloo
sfischme@uwaterloo.ca
Motivation

• **Embedded => Interrupts**
  (Especially in low power systems)
Motivation

• **Embedded** => **Interrupts**
  (Especially in low power systems)

• **Interrupts** => **Complexity**
  (Non-linear program execution)
• **Embedded** => **Interrupts**  
  (Especially in low power systems)

• **Interrupts** => **Complexity**  
  (Non-linear program execution)

• **Complexity** => **Nasty bugs**  
  (You only need one try, right?)
Motivation

$ echo 'int main() { printf ("Hello, world\n"); }' | gcc -xc -o out
Motivation

• Embedded => Interrupts
  (Especially in low power systems)

• Interrupts => Complexity
  (Non-linear program execution)

• Complexity => Nasty bugs
  (You only need one try, right?)

• Nasty bugs require good debugging support
Next Gen Debugging Support?
Next Gen Debugging Support?
Next Gen Debugging Support?
Replay Debugging

Recording at run time  Replay offline (sim)
Replay Debugging

Recording at run time

- Execute instruction
  - No
  - IRQ?
    - Yes
      - Log IRQ (fingerprint)
      - Execute ISR & return
    - No
      - Execute instruction

Replay offline (sim)
Replay Debugging

Recording at run time

1. Execute instruction

   - No
     - IRQ?
       - Yes
         - Log IRQ (fingerprint)
         - Execute ISR & return
       - No
         - IRQ?

Replay offline (sim)

1. Simulate instruction

   - No
     - Calculate fingerprint
     - in DB?
       - Yes
         - Simulate ISR
Replay Debugging

<table>
<thead>
<tr>
<th>Recording at run time</th>
<th>Replay offline (sim)</th>
</tr>
</thead>
</table>

Non-determinism happens (e.g., interrupt)
# Replay Debugging

<table>
<thead>
<tr>
<th>Recording at run time</th>
<th>Replay offline (sim)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task A</td>
<td></td>
</tr>
</tbody>
</table>

- Non-determinism happens (e.g., interrupt)
Replay Debugging

Recording at run time

Replay offline (sim)

Task A

Non-determinism happens (e.g., interrupt)
Replay Debugging

Recording at run time

Replay offline (sim)

Task A

ISR

× Non-determinism happens (e.g., interrupt)
Replay Debugging

Recording at run time

Replay offline (sim)

Task A

ISR

Store fingerprint

DB

Non-determinism happens (e.g., interrupt)
Replay Debugging

Recording at run time

Replay offline (sim)

Task A

ISR

Store fingerprint

DB

Non-determinism happens (e.g., interrupt)
Replay Debugging

Recording at run time

Task A

ISR

Store fingerprint

DB

Replay offline (sim)

Task A

ISR

Non-determinism happens (e.g., interrupt)
Replay Debugging

Recording at run time

Task A

ISR

Store fingerprint

DB

Replay offline (sim)

Task A

ISR

Non-determinism happens (e.g., interrupt)

Sunday, June 21, 2009
Replay Debugging

Non-determinism happens (e.g., interrupt)
Replay Debugging

Recording at run time

Task A

ISR

Store fingerprint

DB

Task A

ISR

Replay offline (sim)

Non-determinism happens (e.g., interrupt)

Non-determinism happens (e.g., interrupt)
Non-determinism happens (e.g., interrupt)
Replay Debugging

Recording at run time

Task A

ISR

Store fingerprint

DB

Replay offline (sim)

Task A

ISR

Non-determinism happens (e.g., interrupt)
Non-determinism happens (e.g., interrupt)
Replay Debugging

Recording at run time

Task A

ISR

Store fingerprint

DB

Replay offline (sim)

Task A

ISR

X

X

X

Non-determinism happens (e.g., interrupt)
Replay Debugging

Recording at run time

Replay offline (sim)

Non-determinism happens (e.g., interrupt)
Replay Debugging

Recording at run time

Task A

ISR

Store fingerprint

DB

Replay offline (sim)

Task A

ISR

Non-determinism happens (e.g., interrupt)
Benefits

- Communicate and document bugs
  (Show the trace)

- Go forward and backwards in time
  (Thoroughly analyze how the bug happened)

- Retest specific scenario for fixes
  (Exercise the system with the specific trace)
Non-determinism happens (e.g., interrupt)
Recording Phase

First try: record program counter

Non-determinism happens (e.g., interrupt)
Non-determinism happens (e.g., interrupt)
Recording Phase

First try: record program counter

Second try: record program counter and state information

Non-determinism happens (e.g., interrupt)
Recording Phase

First try: record program counter

Second try: record program counter and state information

Non-determinism happens (e.g., interrupt)
Fingerprinting

Compress system state

System state 512+ bits

PC R1 R2
S 0x.... Rx
I/O Rx

Fingerprint <32 bits
01001000100100100100100101110
System Model

Execution context $s$

Selector function $g$

Input components $c$

Hash function $h$

Raw fingerprint $f^*$

Return address $ra$

Extraction of $ra$

Reduction function $r$

Reduced RA $ra^*$

Fingerprint $f$

$\langle ra^*, f^* \rangle$
Caveats

- **False input duplicates** *(selected bad subset)*

  \[ s \neq s' \land g(s) = g(s') \]

- **False loop positives** *(falsely believe IRQ happened here)*

  \[ s \neq s' \land s.ra = s'.ra \text{ but also } (h \cdot g)(s) = (h \cdot g)(s') \]

- **False RA positives** *(falsely believe IRQ happened here)*

  \[ s \neq s' \land s.ra \neq s'.ra \text{ but also } r(s.ra) = r(s'.ra) \land (h \cdot g)(s) = (h \cdot g)(s') \]
Caveats

- False input duplicates \((\text{selected \ bad \ subset})\)
  \[ s \neq s' \land g(s) = g(s') \]

- False loop positives \((\text{falsely believe IRQ happened here})\)
  \[ s \neq s' \land s.ra = s'.ra \text{ but also } (h \cdot g)(s) = (h \cdot g)(s') \]

- False RA positives \((\text{falsely believe IRQ happened here})\)
  \[ s \neq s' \land s.ra \neq s'.ra \text{ but also } r(s.ra) = r(s'.ra) \land (h \cdot g)(s) = (h \cdot g)(s') \]

Calls for good design of fingerprint
Designing the Fingerprint

1. Raw fingerprint $f^*$
2. Hash function $h$
3. Selector function $g$
4. Execution context $s$
5. Return address $ra$
6. Reduction function $r$

Result: Fingerprint $f$ with components $\langle ra^*, f^* \rangle$
Execution context $s$

Selector function $g$

Input components $c$

Hash function $h$

Raw fingerprint $f^*$

Extraction of $ra$

Return address $ra$

Reduction function $r$

Reduced RA $ra^*$

Fingerprint $f$

$\langle ra^*, f^* \rangle$
• **Many elements to choose from:** data registers, control registers, GPIO, IO status, general SDRAM, stack, heap, ...
• **Many elements to choose from:** data registers, control registers, GPIO, IO status, general SDRAM, stack, heap, ...

• **Can’t take all:**
  • Too much computation overhead
  • Not all are used equally often
Idea 1

Pick the most frequently used ones
Idea 1

Pick the most frequently used ones

Register Usage (freq. > 0.012)

Relative frequency

Register

Sunday, June 21, 2009
Does it work?

Yes, quite well.

ADC Application
Percentage of false input duplicates

Including $x$ most frequently used registers
Does it work?

ADC Application

FuncsHeap Application

FuncsHeapWithoutLoop Application

Sunday, June 21, 2009
Execution context $s$

Selector function $g$

Input components $c$

Hash function $h$

Raw fingerprint $f^*$

Fingerprint $f$

$\langle ra^*, f^* \rangle$

Extraction of $ra$

Return address $ra$

Reduction function $r$

Reduced RA $ra^*$
Comparison of the Hash Functions

- Median count of hash collisions

- Non-optimal selector function
- Non-optimal selector function with RR hashing
- Optimal selector function

Width of $f^*$

Sunday, June 21, 2009
Return addresses tend to cluster

Clustering of Interrupt Return Addresses

Return addresses tend to cluster
Return addresses tend to cluster
Return addresses tend to cluster

- **Tight loops**: busy waiting, iterative calculations, array/matrix operations
Why?

Return addresses tend to cluster

• **Tight loops**: busy waiting, iterative calculations, array/matrix operations

• **Blocking operations**: non-preemptive peripheral access, waiting for user input, costly instructions
Return addresses tend to cluster

- **Tight loops**: busy waiting, iterative calculations, array/matrix operations

- **Blocking operations**: non-preemptive peripheral access, waiting for user input, costly instructions

- **Disabled interrupts**: generic concurrency control, I/O access
Why?

Return addresses tend to cluster

- **Tight loops**: busy waiting, iterative calculations, array/matrix operations
- **Blocking operations**: non-preemptive peripheral access, waiting for user input, costly instructions
- **Disabled interrupts**: generic concurrency control, I/O access

Context specific hashing => RR hashing
Idea 2: Round-Robin Hashing
Idea 2: Round-Robin Hashing

- Use **different hash functions** to minimize collisions
Idea 2: Round-Robin Hashing

- Use **different hash functions** to minimize collisions
- Insert **markers** in the code for different hash functions
Idea 2: Round-Robin Hashing

• Use **different hash functions** to minimize collisions

• Insert **markers** in the code for different hash functions

• In the replay use the correct hash function for according to the marker
RR hashing works surprisingly well

Comparison of the Hash Functions

- Optimal selector function
- Non-optimal selector function with RR hashing
- Non-optimal selector function

Median count of hash collisions vs Width of $f^*$

Sunday, June 21, 2009
Execution context $s$

Selector function $g$

Input components $c$

Hash function $h$

Raw fingerprint $f^*$

Return address $ra$

Extraction of $ra$

Reduction function $r$

Reduced RA $ra^*$

Fingerprint $f$

$\langle ra^*, f^* \rangle$
How best to use the FP bits?

- Give all to the hash value? \textit{(minimize hash collisions)}
- Give all to the RA? \textit{(minimize false RA positives)}
Trade-off Between $ra^*$ and $f^*$ Width

- **RA positives**
- **Hash collisions**

Optimal fingerprint partitioning

Frequency [%] vs. Width of $ra^*$
Conclusions

• Debugging embedded systems is painful
  => Replay debugging can help [ST08]

• Frequency-based selection function ✔
• Round robin hashing ✔
• Tradeoff engineering for RA vs hash ✔

• Future work: Lots, see discussion section
Questions?

Work has been sponsored by:
Graduate Students' Exchange Program (GSEP) funded by Foreign Affairs and International Trade Canada (DFAIT),
Ontario Research Fund Research Excellence (ORF-RE) &
CIMIT under U.S. Army Medical Research Acquisition Activity Cooperative Agreement W81XWH-07-2-0011