Flash Memory in Embedded Java Programs

Stephan Korsholm
VIA University College
Horsens, Denmark
sek@viauc.dk

JTRES'2011, York, UK
Overview

• Presentation, 20 min
  - Background
  - What is constant data, and why keep it in Flash?
  - Constant data in a JVM
  - Marking and initialization of constant data
  - Accessing constant data
  - Cost of constant data
  - Future work
  - Perspective

• Discussion, 10 min
Background

• Java on small embedded devices
  - > 256 Kb flash
  - < 8 Kb RAM
  - 8/16/32 bit architecture
  - Usually programmed by engineers familiar with C
What is constant data?

<table>
<thead>
<tr>
<th></th>
<th>Code</th>
<th>Variable Data</th>
<th>Constant Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECT</td>
<td>234KB</td>
<td>7KB (8%)</td>
<td>78KB (92%)</td>
</tr>
<tr>
<td>Modbus</td>
<td>171KB</td>
<td>207KB (83%)</td>
<td>41KB (17%)</td>
</tr>
<tr>
<td>HVM</td>
<td>22KB</td>
<td>4KB (29%)</td>
<td>10KB (71%)</td>
</tr>
</tbody>
</table>

Table 1: Overview of data usage
Why keep it in flash?

- Grundfos devices
  - Circulation pumps in households
- Polycom devices
  - Wireless DECT handsets
  - Conference systems (formula 1, tall ships racing)

Sold in large quantities

Competitive market

Flash is cheaper
Constant data in a JVM
Constant data in a JVM
Constant data in a JVM

• The HVM is a lean Java VM

1. Intelligent class linking
2. SDK independence
3. OS independence
4. Simple build procedure (gcc -nostdlib *.c)
5. HW Objects & 1st level interrupt handling
6. ROM/RAM aware
Constant data in a JVM

• The HVM is a lean Java VM

1. Intelligent class linking
2. SDK independence
3. OS independence
4. Simple build procedure (gcc -nostdlib *.c)
5. HW Objects & 1st level interrupt handling
6. ROM/RAM aware
7. Supports constant data in flash
Marking constant data

@Flash
private int[] array = { 23, 112, -1, -1};

• Explicit marking of constant data
• Using annotations
• Same as in C environments
• Room for improvements
public class ConstantData extends Object{
public ConstantData();

    Code:
    0:  aload 0
    1:  invokespecial #1; //"<init>"():V
    4:  aload 0
    5:  iconst_4
    6:  newarray byte
    8:  dup
    9:  iconst_0
   10:  bipush 23
   11:  bastore
   12:  dup
   13:  iconst_1
   14:  bipush 112
   15:  bastore
   16:  dup
   17:  iconst_2
   18:  bipush -1
   19:  bastore
   20:  dup
   21:  iconst_3
   22:  bipush -1
   23:  bastore
   24:  putfield #2; //wav_num_0:[B
   25:  return
    }

private int[] array = { 23, 112, -1, -1};


Accessing constant data

```java
public class ConstantData extends Object {
    public ConstantData();
    Code:
    0:  aload_0
    1:  invokespecial #1; //"<init>":()V
    4:  aload_0
    5:  iconstant_4
    6:  newarray byte
    8:  dup
    9:  iconstant_0
   10:  bipush     23
   12:  bastore    23
   13:  dup
   14:  iconstant_1
   15:  bipush     112
   17:  bastore    112
   18:  dup
   19:  iconstant_2
   20:  iconstant_m1
   21:  bastore    
   22:  dup
   23:  iconstant_3
   24:  iconstant_m1
   25:  bastore    #2; //wav_num_0:[B
   28:  return
}
```

Host handling

Create in constant heap

No changes on host

Because this field is marked as constant!
package test.icecapvm.minitests;
public class TestVolatile4 {
    private static class ConstantData {
        public volatile int NUM1 = 42;
        public volatile byte[] bytes = { 23, 112, -1, -1 };
        public volatile int NUM2 = 43;
    }

    public static void main(String[] args) {
        ConstantData cdata = new ConstantData();
        devices.System.lockROM();
        if (cdata.NUM1 == 42) {
            if (cdata.bytes != null) {
                if (cdata.bytes.length == 4) {
                    int sum = 0;
                    for (int i = 0; i < 4; i++) {
                        sum += cdata.bytes[i];
                    }
                    if (sum == 133) {
                        if (cdata.NUM2 == 43) {
                            args = null;
                        }
                    }
                }
            }
        }
    }
}
After initialization

#include "types.h"

unsigned char pheap[29] PROGMEM = {
  0x0, 0x0, 0x0, 0x18, 0x5, 0x0, 0x0, 0x0, 0x2a,
  0x0, 0x0, 0x0, 0x26, 0x0, 0x0, 0x0, 0x2b,
  0x0, 0x0, 0xf8, 0x2, 0x0, 0x0, 0x0, 0x4,
  0x0, 0x1, 0x2, 0x3
};

unsigned char rom_writeable(void)
{
  return 0;
}
Architecture
Accessing constant data

```java
public class ConstantData extends Object{
    public ConstantData();
    Code:
    0:  aload_0
    1:  invokespecial #1; //"<init>"():V
    4:  aload_0
    5:  icrostant_4
    6:  newarray byte
    8:  dup
    9:  icrostant_0
   10:  bipush   23
   11:  icrostant_3
   12:  bipush   112
   13:  dup
   14:  icrostant_1
   15:  bipush   23
   16:  icrostant_4
   17:  bipush   112
   18:  dup
   19:  icrostant_0
   20:  bipush   -1
   21:  icrostant_3
   22:  bipush   -1
   23:  icrostant_1
   24:  bipush   #2; //wav_num_0:[B
   25:  icrostant_4
   26:  bipush   0
   27:  putfield #2; //wav_num_0:[B
   28:  return
}
```

Target handling

Don't create in constant heap (it's already there)
Just return reference to it

No changes
Accessing constant data

• Arrays
  - newarray
    must know if it is a constant array!
  - array load, array store
    must check if it is a constant array
Harvard vs. Von Neumann

• Objects
  - new
    must know if it contains constant fields!
  - get field, put field
    must know if it contains constant fields
Harvard vs. Von Neumann again
  - instanceof, checkcast

Target handling

public class ConstantData extends Object{
  public ConstantData();
  Code:
  0:  aload_0
  1:  invokespecial #1; //"<init>":(V
  4:  aload_0
  5:  iconstant_4
  6:  newarray byte
  8:  dup
  9:  iconstant_0  
 10:  bipush   23
 12:  bastore
 13:  dup
 14:  iconstant_1
 15:  bipush   112
 17:  bastore
 18:  dup
 19:  iconstant_2
 20:  iconstant_m1
 21:  bastore
 22:  dup
 23:  iconstant_3
 24:  iconstant_m1
 25:  bastore
 26:  putfield #2; //wav_num_0:[B
 29:  return
}
Cost of constant data

- If I don't use constant data what is the cost?

- If I do use constant data what is the cost for accessing that data?
Cost of constant data

• If I don't use constant data what is the cost? 7.5%

• If I do use constant data what is the cost for accessing that data? 33%
Conclusion

• Two industrial applications could not run on a JVM on the target
• We added the option for constant data in flash
• Now they may be ported
• Currently we are porting the Grundfos application to Java
Making it better

• Implicit marking of constant data

• Implicit marking of initialization phase

• Avoiding cost when not using constant data
Perspective

• Well known features from C adopted to the Java domain,
  - Hardware Objects (Schoeberl)
  - 1st level interrupt handlers
  - Constant data in flash

• Next steps,
  - Debugging devices using Eclipse
Questions/comments?