Energy Awareness for Embedded Systems

OPTIMIZING EMBEDDED SOFTWARE FOR POWER
Introduction

• Review of Power Consumption
• Understanding Power for Embedded Systems
• Software and Hardware Optimizations
Review of Power Consumption

1. Application
2. Technology
3. Voltage
4. Frequency
Review of Power Consumption

\[ P_{\text{total}} = P_{\text{Dynamic}} + P_{\text{Static}} \]

Types of Power:

- Maximum
- Average
- Worst-Case
- Typical
Minimizing Power Consumption

1. Hardware Techniques
2. Data Flow Optimization
3. Algorithmic Optimization
Hardware Techniques (1)

Low Power Modes

- Power gating
- Clock gating
- Voltage Control
- Frequency Control
Hardware Techniques (2)

Considerations

Available Block Functionality
• Memory states and validity must be considered
• Certain peripherals will not be available

Overhead
• Not break real-time constraints
Data Flow Optimization—Memory Access (1)

Principle of locality

When the CPU needs data, it looks first in cache memory.

STEP 1  STEP 2  STEP 3  STEP 4
Data Flow Optimization—Memory Access (2)

Interleaving
Data Flow Optimization—Memory Access (3)

Burst Access

<table>
<thead>
<tr>
<th>Address</th>
<th>Byte 0</th>
<th>Byte 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16,777,712</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Data Flow Optimization—Memory Access (4)
Compiler cache optimizations

Array Merging

```c
int array1[ array_size ];
int array2[ array_size ];

{ 
    int array1;
    int array2;
} new_array[ array_ size ]
```

Loop Interchanging

```c
for (i = 0; i<100; i = i + 1)
    for (j = 0; j<200; j = j + 1)
        for (k = 0; k<10000; k = k + 1)
            z[ k ][ j ] = 10 * z[ k ][ j ];

for (i = 0; i<100; i = i + 1)
    for (k = 0; k<10000; k = k + 1)
        for (j = 0; j<200; j = j + 1)
            z[ k ][ j ] = 10 * z[ k ][ j ];
```
Data Flow Optimization – Peripherals

- **Coprocessors**
  - DMA

- **Bus Configuration**

- **Core Communication**
  - Polling
  - Time-Based Processing
  - Interrupt Processing
Algorithmic Optimization (1)

Loop Unrolling

Regular loop:
for (i = 0; i < 100; i = i + 1)
for (k = 0; k < 10000; k = k + 1)
a[i] = 10 * b[k];

Loop unrolled by 4x:
for (i = 0; i < 100; i = i + 4)
for (k = 0; k < 10000; k = k + 4)
{
    a[i] = 10 * b[k];
    a[i + 1] = 10 * b[k + 1];
    a[i + 2] = 10 * b[k + 2];
    a[i + 3] = 10 * b[k + 3];
}
Algorithmic Optimization (2)

Software Pipelining

Regular Loop:
for (i = 0; i < 100; i = i + 1) {
  a[i] = 10 * b[i];
  b[i] = 10 * c[i];
  c[i] = 10 * d[i];
}

for (i = 0; i < 100 - 2; i = i + 1) {
  c[i] = 10 * d[i];
  b[i + 1] = 10 * c[i + 1];
  a[i + 2] = 10 * b[i + 2];
}
Algorithmic Optimization (3)

Eliminating Recursion

\( fn!(0) = 1 \) For \( n == 0 \)
\( fn!(n) = fn!(n - 1); \) For \( n > 0 \)

```c
int res = 1;
for(int i = 0; i < n; i++)
{
  res* = i;
}
```
Algorithmic Optimization (4)

Reducing Accuracy

Low Power Code Sequences
Algorithmic Optimization (5)

OptAlg

- Tool that automates the optimization of power-intensive algorithmic constructs using symbolic algebra with energy profiling

\[
\text{for } i=1..3 \\
y = y + \cos(i \times x); \\
y = \cos(x) + \cos(2 \times x) + \cos(3 \times x);
\]

\[
y = 1 - \frac{1}{2}x^2 + \frac{1}{24}x^4 + 1 - \frac{1}{2}x^2 + \frac{1}{24}x^4 + 1 - \frac{1}{2}3^2x^2 + \frac{1}{24}3^4x^4 \\
y = 3 - 7x^2 + \frac{49}{12}x^4 \\
y = 3 + (-7 + \frac{49}{12}x^2)x^2
\]
Algorithmic Optimization (6)

OptAlg Flow

1. Algorithmic-level C Code
2. Data Representation Conversion
3. Energy Profiling
4. Critical Basic Blocks
5. Polynomial Formulation
6. Symbolic Algebra Decomposition
7. Optimization Done?
8. Optimized C Code with inline Assembly
# Architecture Level Optimization (1)

<table>
<thead>
<tr>
<th></th>
<th>GPP – based solutions (General Purpose Processor)</th>
<th>ASIC – based solutions (Custom silicon)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pro</td>
<td>• Programmability: flexible</td>
<td>• Very fast</td>
</tr>
<tr>
<td></td>
<td>• Component reuse</td>
<td>• Highly scalable</td>
</tr>
<tr>
<td></td>
<td>• Relatively low cost (RISC)</td>
<td></td>
</tr>
<tr>
<td>Con</td>
<td>• Low performance</td>
<td>• Long time to market</td>
</tr>
<tr>
<td></td>
<td>• Power requirements</td>
<td>• High cost</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Inflexible: little reuse</td>
</tr>
</tbody>
</table>
Architecture Level Optimization (2)

Clustered Length-Adaptive Word Processor (CLAW)

- Allows dynamic modification of the issue width
References

1) Length Adaptive Processors: A Solution for the Energy/Performance Dilemma in Embedded Systems, Iyer, Conte, School of Computer Science, College of Computing, Georgia Institute of Technology, Atlanta, GA

2) Low Power Embedded Software Optimization using Symbolic Algebra, Peymandoust, Simunic, De Micheli, Stanford University
Questions?