Lightweight Semantics-Preserving Communication for Real-Time Automotive Software

Research support was partially provided by the Bayerische Forschungsstiftung under grant no. AZ-1257-16

Eugene Yip  Erjola Lalo  Gerald Lüttgen  Andreas Sailer

International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)
4 October 2019
Automotive Applications on Multi-Cores

• **Real-time and safety-critical software**
  – Correct outputs required at correct times!
  – AUTomotive Open System ARchitecture (AUTOSAR) for designing software applications, later implemented as tasks

• **Multi-core challenge**
  – Need to deploy *legacy single-core applications* on multi-cores, alongside modern applications
  – *Maintain system’s timing behaviour* when exploring deployment options
  – *Time predictable and deterministic communication needed*
Logical Execution Time (LET) Model

- Automotive industry tackling the multi-core challenge with *LET*
  - *Timing contract* between system designers, control engineers, and software developers

- Time-predictable and deterministic execution behaviour
  - Implementor’s responsibility to preserve the LET timing model
  - *Invariant to platform changes*

- *Buffering* needed for task communication
Real-Time Buffering Protocols

- **Point-to-Point (PTP)** protocol used in automotive LET systems
  - Statically managed, relies on high-priority LET drivers, memory inefficient, and high bus traffic at LET boundaries
  
  [Resmerita et al. Efficient Realization of Logical Execution Times in Legacy Embedded Software. 2017]

- **Dynamic Buffering Protocol (DBP)** for synchronous programs
  - Tasks assumed to execute and communicate in zero time
  - Dynamically managed, semantics preserving, memory optimal, relies on fixed-priority scheduling, and suited to single-core


- **Contributions**: Lightweight buffering protocol for LET systems
  - Extend DBP: LET semantics, multiple writers, amenable to multi-cores
  - Reduce: Buffer sizes, run-time overheads, and frequency of buffer writes
Main Assumptions

- LET task set, i.e., grouping of functions into schedulable entities, is given
- Tasks do not make local copies of signals

\[ \text{LET} \leq \text{Period} \]

\[ \text{Period} \]

\[ t_x \]

\[ t_y \]

\[ t_z \]

© 2019 E. Yip, E. Lalo, G. Lütten, A. Sailer

Lightweight Semantics-Preserving Communication for Real-Time Automotive Software – 5
Static Buffering Protocol

- SBP analyses each signal separately over one hyper-period
  - Least common multiple of task periods
  - One buffer per signal, and initial value is in element $e_0$
  - Buffer elements reserved for the entire LET
  - Buffering actions saved in a buffering schedule
Static Buffering Protocol

- **Time 0ms**
  1. Readers $t_b$ and $t_c$ start: Read initial value. Occupy element $e_0$
  2. Writer $t_a$ starts: **Buffer is full**. Occupy new element $e_1$

- **Time 0.5ms**
  1. Reader $t_b$ ends: Release $e_0$ (but still occupied by $t_c$)
  2. Writer $t_a$ ends: Write value to $e_1$ and release $e_1$
Static Buffering Protocol

- **Time 1ms**
  - Writer $t_a$ starts: Reuse $e_1$

- **Time 1.5ms**
  - Writer $t_a$ ends: Write value to $e_1$ and release $e_1$
Static Buffering Protocol

- **Time 2ms**
  1. Reader $t_c$ ends: Release $e_0$ (now unoccupied)
  2. Reader $t_b$ starts: Read latest value. Occupy $e_1$
  3. Writer $t_a$ starts: Reuse $e_0$

- **Time 2.5ms**
  1. Reader $t_b$ ends: Release $e_1$
  2. Writer $t_a$ ends: Write value to $e_0$ and release $e_0$
Static Buffering Protocol

- **Fast forward to time 4ms**
  1. Reader $t_b$ starts: Read latest value. Occupy $e_1$
  2. Writer $t_a$ starts: **Buffer is full**. Occupy new $e_2$

- **Fast forward to time 6ms**
  - Last writer instance needed to initialise the signal for the next hyper-period
  - Routine needed to copy $e_2$ into $e_0$
Evaluation

- Evaluated the memory consumption and execution overhead of SBP against traditional PTP protocol
  - **PTP:** Created drivers for reads/writes at LET boundaries
  - **SBP:** Replaced signal accesses with buffer (array) accesses
  - Hardware model was *AURIX TC27x* automotive tri-core processor

- Automotive benchmarks
  - *Synthetic models* parameterised from actual automotive applications
  - *Industrial model* of an engine management system from Bosch [Hamann et al. *WATERS Industrial Challenge 2017.*]
  - 10’s of tasks, 1000’s of signals, 10,000’s of accesses
  - *Stack/local variables* of a task allocated to their core’s local memory
  - *Signals and global buffers* allocated to the global memory
### Evaluation

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Buffer Memory (\text{(plus auxiliary memory)})</th>
<th>Execution Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTP</td>
<td>Highest (\text{(no auxiliary memory needed)})</td>
<td>Highest</td>
</tr>
<tr>
<td></td>
<td>• Scheduling of LET drivers</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Serialisation of drivers and bus contention at LET boundaries</td>
<td></td>
</tr>
<tr>
<td>SBP</td>
<td>40% (\text{(70%)}) of PTP</td>
<td>20% of PTP</td>
</tr>
<tr>
<td></td>
<td>• Buffer indexes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Updating of buffer indexes</td>
<td></td>
</tr>
</tbody>
</table>

- Auxiliary memory is freed upon task termination
- **SBP is a good compromise between memory and overhead**
  - Must also consider implementation effort
  - **PTP:** Create and schedule LET drivers
  - **SBP:** Modify source code for buffered accesses.
    Regenerate the buffering schedule on design change
Conclusions

• **SBP**: Extended DBP to comply with LET semantics, and to support multiple writers on multi-cores

• **SBP is a good compromise** between buffer memory and execution overhead compared to PTP
  – *Source code has to be modified*

• SBP can be improved by *suppressing unnecessary writes*
  – Requires intermediate output values to be stored locally, and data age constraints on inputs
  – Heuristic for finding a subset of writes that satisfy the readers
  – *20% the buffer memory of PTP*, but *50% the execution overhead of PTP*
Thank you!

Questions?