Preliminary design and validation of a modular framework for predictable composition of medical imaging applications

7th July 2015

Image display

- An input signal needs several transformations before being displayed
Envisioned new design

Main modifications

- Replace proprietary FPGAs by a COTS platforms
- Shift to a component-based software architecture

![Diagram showing the transition from FPGA to Software Component Framework (tools + run-time environment) with IMX 6 and Atom N270 as examples.]
Envisioned new design

Main modifications

- Replace proprietary FPGAs by a COTS platforms
- Shift to a component-based software architecture

Advantages

- Support for product variants
- Time-to-market:
  - Independent development & testing of components
Problem Description

Key issues for development

- Resources
- Performance requirements
- Desired functionality

\{ Manage variations at design time \}
Problem Description

Key issues for development

- Resources
- Performance requirements
- Desired functionality

\[
\begin{align*}
\text{Manage variations at design time}
\end{align*}
\]

Goals:

- COTS software framework
- Predictable framework configuration and performance metrics
- Validate predicted performance against run-time performance

Build a prototype!
COTS Software – Logical View

**Software**
- Define interface in QML language
- Define pipeline architecture
- Proprietary algorithms for various signal transformations

**Hardware**
- Various COTS platforms

**Software**
- User Interface
- Image Processing Pipeline
- Custom Algorithms

**Hardware**
- IMX 6
- Atom N270
COTS Software – new interfaces

- Plugin for integrating Gstreamer into QML language => high level development
- Integrate Matlab code into Gstreamer => high reusability
Adding Performance Analysis

- User Interface
- Image Processing Pipeline
- C Wrapper for Matlab
- Custom Algorithms
- IMX 6
- Atom N270

Requirements (throughput)

Performance met? (Y/N)

Expected performance
Adding Performance Analysis and variability management

User Interface

Image Processing Pipeline

C Wrapper for Matlab

Custom Algorithms

Pipeline Configuration PC1... PC2 ...

Performance Analysis

Performance met? (Y/N)

Expected performance

Requirements (throughput) R

IMX 6 Atom N270...
Adding Performance Analysis and variability management

- User Interface
- qt-quickstreamer
- Image Processing Pipeline
- C Wrapper for Matlab
- Custom Algorithms

Pipeline Configuration: PC1... PC2 ...

Performance Analysis

Performance met? (Y/N)
Expected performance

Requirements (throughput) R

- IMX 6
- Atom N270
- ...
Use case: applying the concepts

Input video stream formats to be supported
- SD - 576i: MPEG2 Transport Stream up to 6 Mbit/s
- HD - 720p: H.264 Transport Stream up to 15 Mbit/s
- HD - 1080i: H.264 Transport Stream up to 20 Mbit/s

Input video transport formats to be supported
- Multicast UDP
- RTP

Output screen resolutions
- 1366x768
- 1920x1080

Hardware
- Intel Atom N270
- Intel Cedarview D2550
- Freescale iMX6 dual and quad core
- Intel Baytrail DN2820
Use case: Requirements analysis

Input video stream formats to be supported:
- SD - 576i: MPEG2 Transport Stream up to 6 Mbit/s
- HD - 720p: H.264 Transport Stream up to 15 Mbit/s
- HD - 1080i: H.264 Transport Stream up to 20 Mbit/s

Input video transport formats to be supported:
- Multicast UDP
- RTP

Output screen resolutions:
- 1366x768
- 1920x1080

Hardware:
- Intel Atom N270
- Intel Cedarview D2550
- Freescale iMX6 dual and quad core
- Intel Baytrail DN2820
Use case: variability

Input video stream formats to be supported
SD - 576i: MPEG2 Transport Stream up to 6 Mbit/s
HD - 720p: H.264 Transport Stream up to 15 Mbit/s
HD - 1080i: H.264 Transport Stream up to 20 Mbit/s

Input video transport formats to be supported
Multicast UDP
RTP

Output screen resolutions
1366x768
1920x1080

Hardware
Intel Atom N270
Intel Cedarview D2550
Freescale iMX6 dual and quad core
Intel Baytrail DN2820
Research questions

1. How to model GStreamer pipelines?

2. What are the required inputs for performance analysis?

3. What is the mapping between an GStreamer pipeline and a performance model?

4. What are the key configuration parameters of a GStreamer pipeline?
GStreamer Pipeline Architecture

- A number of plugins can be connected to attain the requisite media processing
- The processing unit in GStreamer is called a pipeline
- It handles the clocking, the synchronizations, scheduling and the control message flow between elements
Important GStreamer elements

Buffers
• Media content passed between elements
• May have different sizes

Queues
• Represent thread boundaries
• Enable/disable back pressure (i.e., write protection)
# Mapping of Gstreamer and Synchronous Data-Flow models

<table>
<thead>
<tr>
<th>SDF element</th>
<th>Gstreamer equivalent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>Set of linked elements running on same thread</td>
<td>Functionality, code to be executed</td>
</tr>
<tr>
<td>Token</td>
<td>Buffer</td>
<td>Data units</td>
</tr>
<tr>
<td>Channels</td>
<td>Pad links</td>
<td>Data dependencies/execution order</td>
</tr>
<tr>
<td>Rate</td>
<td>#buffers pushed/popped</td>
<td>Data units consumed/produced</td>
</tr>
</tbody>
</table>

![Diagram of Gstreamer and Synchronous Data-Flow models](image)
Traffic shaper:
- ensures that a data unit is equal to a video frame;
- If no back-pressure (data may be overwritten), then limit data rate of the source
Protect overwrites with back pressure

Buffer sizing vs. performance
No back pressure

Largest buffer sizing!

Data loss vs. highest throughput:
Performance analysis vs. measures
(with vs. without back pressure)

<table>
<thead>
<tr>
<th>Back Pressure</th>
<th>Distribution</th>
<th>Worst case Run-time</th>
<th>Predicted Throughput (fps)</th>
<th>Average Run-time Throughput (fps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>(2,1,1)</td>
<td>(1,1,1)</td>
<td>28</td>
<td>31</td>
</tr>
<tr>
<td>Disabled</td>
<td>(2,2,2)</td>
<td>(1,1,2)</td>
<td>31</td>
<td>31</td>
</tr>
</tbody>
</table>

**Diagram:**
- File source
- Traffic Shaper
- Filter
- x264 dec
- Csp
- Scaler
- Sink

Actor A ➔ Q1(AB) ➔ Actor B ➔ Q2 (BC) ➔ Actor C ➔ Q3 (CD) ➔ Actor D
Run-time analysis of memory usage (back-pressure enabled)

- Run-time monitoring of push/pop events on buffers
- Visualization using Time Doctor (http://sourceforge.net/projects/timedoctor/)
Run-time analysis of memory usage (back-pressure disabled)

- Run-time monitoring of push/pop events on buffers
- Visualization using Time Doctor (http://sourceforge.net/projects/timedoctor/)
Conclusions

Current work
- Investigate
  - COTS software framework
  - Predictable framework configuration and performance metrics
- Prototyping:
  - predicted performance against run-time performance

Future work
- Complex pipelines (split and joins)
- GStreamer scalability
- Advanced platform models (processor mappings, caches, etc.)