LibGALS: A Library for GALS Systems
Design and Modeling

Wei-Tsun Sun*, Zoran Salcic and Avinash Malik

Department of Electrical and Computer Engineering
University of Auckland
New Zealand

Email : wsun013@aucklanduni.ac.nz*

January 18 2009
What is LibGALS?

- A library implemented based on C to support GALS MoC
- GALS – Globally Asynchronous Locally Synchronous
- Provide programming interface to describe GALS Systems
- Programming constructs are intuitive – no low-level details on communications and synchronizations
- Not only for single processor systems but also for multi-core/multi-processing architectures
A GALS software system (program)

- A system can be described and modeled by a number of concurrent sequential behaviors.

- Behaviors can run at the same pace or different speeds.

- Behaviors require communications and synchronizations:
  - Communications used to exchange information.
  - Synchronizations to maintain integrity of the concurrency.
What is a GALS system in a LibGALS program?

A program that describes a GALS system using LibGALS is called a **LibGALS program**.

Entities of a LibGALS program consists of basic building blocks including:
- Clock domains – asynchronous behaviors
- Reactions – synchronous behaviors
- Channels – communication between asynchronous behaviors
- Signals – communication between synchronous behaviors
What is a GALS system in a LibGALS program?

A GALS system is described with these building blocks:

- **Environment**: Other software components of the system
- **Inputs from the Environment**
- **Outputs to the Environment**
- **Reaction R1**
- **Reaction R2**
- **Reaction R3**
- **Signal**
- **Channel**
- **Synchronizer process**

The diagram shows the flow of information and interactions within a GALS system, including the roles of different components and their connections.
What is a GALS system in a LibGALS program?

- A *clock domain* is an entity in a GALS system which may consist of one or more reactions.
- A GALS program can include one or more clock domains.
- Reactions are acting synchronously:
  - Follows logical ticks – barrier synchronization
  - Allow creation of children reactions
- Clock domains are acting asynchronously to each other.
What is a GALS system in a LibGALS program?

- A reaction can be a composition of other reactions known as *children reactions*.

- As illustrated, reaction R2 and R3 are children reactions of reaction R1.
What is a GALS system in a LibGALS program?

- Reactions are executed in lock-steps called ticks.

- Ticks are logical times, which can be of different length in real time units.

- Reactions of the same clock domain share one common tick; a reaction will be in the same tick as long as all reactions of the same clock domain finish their ticks.
What is a GALS system in a LibGALS program?

- Reactions communicate and synchronize with each other via **signals**

- Two types of signals – **pure signals** and **valued signals**
  - A pure signal can either be present or absent
  - A valued signal is a pure signal with extra value attribute
  - The value of a valued signal is persistent

- Signal can be made present through **emission** by a reaction

- The presence of signal is broadcast within a clock domain at the current tick. The same as Esterel and SystemJ
What is a GALS system in a LibGALS program?

Signals in GALS system

Environment eg. Other software components of the system

Inputs from the Environment

Clock Domain

Reaction R1
Signal
Reaction R2

Reaction R3

Synchronizer process

Outputs to the Environment

Clock Domain

Reaction

Channel

Synchronizer process
What is a GALS system in a LibGALS program?

- A signal is used within a clock domain.
- Signals are also used as inputs and outputs of a clock domain.
  - To sample environment as input to the clock domain.
  - Similar, to generate outputs to the environments.
  - Inputs and outputs are occurred according to ticks.
- Communication between clock domains are via channels.
  - Channels are point-to-point and uni-directional.
  - Data sent by channel are copied instead of shared.
- Channels synchronized by rendezvous as in CSP.
What is a GALS system in a LibGALS program?

Channels in GALS system

Environment (e.g., other software components of the system) → Inputs from the Clock Domain → Outputs to the Environment

Clock Domain

Reaction R1 → Reaction R2 → Reaction R3 → Channel → Reaction

Synchronizer process
How LibGALS is implemented?

- A set of data structures is established to “bookkeep” the status of clock domains, reactions, channels, and signals.

- Clock domains are containers which link with relevant reactions, and signals, and is registered with channels.

- Each reaction is implemented as a process/thread. Pthread is used at the current implementation of the LibGALS.

- To resolve dependencies between reaction processes/threads, a synchronizer process is introduced.

- LibGALS program is multi-thread in nature.
How LibGALS is implemented?

- Each clock domain will have a synchronizer process.
- Synchronizer process is formed automatically with the creation of a clock domain.
- Synchronizer is programmer invisible – abstracts the details out.

- Semaphores are used in libGALS internally as part of its data structure.
- Only process and semaphore operations are essential to every OS and are required by the LibGALS.
- Hence LibGALS is highly portable!
Inputs and outputs of a clock domain

- Inputs and outputs are implemented as functions registered with clock domains
  - They are known as input/output functions
  - They are activated at tick edges

- Inputs and outputs functions are interfaces to other programs and device drivers outside of the LigGALS program
LibGALS is implemented by using OS services
- Reactions and synchronizers are implemented based on LibGALS and other OS services
- Other software communicate with reactions via I/O functions
LibGALS provides a set of application programming interfaces (APIs) which are intuitive and easy to use.

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createClockDomain</td>
<td>Create a clock domain</td>
</tr>
<tr>
<td>createReaction</td>
<td>Create a reaction within a clock domain</td>
</tr>
<tr>
<td>create[Signal</td>
<td>Trap]</td>
</tr>
<tr>
<td>startClockDomain</td>
<td>Start running a clock domain</td>
</tr>
<tr>
<td>initReaction/</td>
<td>Initialize a reaction and end initialization of the reaction</td>
</tr>
<tr>
<td>endinitReaction</td>
<td></td>
</tr>
<tr>
<td>getArgument</td>
<td>Get an argument passed to the reaction</td>
</tr>
<tr>
<td>endReaction</td>
<td>End a reaction, called if the reaction is not a child reaction</td>
</tr>
</tbody>
</table>
# How to use LibGALS

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>register[Emitter</td>
<td>Trap]</td>
</tr>
<tr>
<td>emit</td>
<td>sustain</td>
</tr>
<tr>
<td>present</td>
<td>Check if a signal is present</td>
</tr>
<tr>
<td>pause</td>
<td>Enforce end of tick for a reaction</td>
</tr>
<tr>
<td>await</td>
<td>Wait for the presence of a signal</td>
</tr>
<tr>
<td>[strong</td>
<td>weak] abort/endAbort</td>
</tr>
<tr>
<td>suspend/endSuspend</td>
<td>Suspend a reaction by one tick if a monitored signals are present</td>
</tr>
<tr>
<td>setTrap/endTrap</td>
<td>Set and end the scope of the trap</td>
</tr>
</tbody>
</table>
### How to use LibGALS

<table>
<thead>
<tr>
<th>AND,OR,NOT,REP</th>
<th>Form a combined signal expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Acquire the value of a signal</td>
</tr>
<tr>
<td>pre[Value]</td>
<td>Get the presence status and value of a signal in the previous tick</td>
</tr>
<tr>
<td>createChannel</td>
<td>Create a channel connecting two clock domains</td>
</tr>
<tr>
<td>send/receive</td>
<td>Send and receive data between reactions in different clock domains via a channel</td>
</tr>
</tbody>
</table>
How LibGALS is implemented?

- APIs are implemented by using data structures and basic building blocks of LibGALS.

- For example, abort is implemented with checking presence of *signals* with *goto* statements built-in as *macros*.

- Another example, *traps* behave similarly to signals, whose activation is triggered by mechanism similar to *signal emission*.

- A trap is a variant (more restricted version) of a signal.
Uses of LibGALS

Less than 200 lines of code (and most of them are in this paper) are required to describe a power kite controller system shown below.

Inputs from the Environment:
- WindHeading
- WindSpeed

Environment (e.g., Other software components of the system):
- ReadWindSpeed
- SendWindData
- PutWindSpeed

Outputs to the Environment:
- ReduceKiteBearing
- ReduceKiteVelocity
- IncreaseKiteBearing
- IncreaseKiteVelocity

Environment (e.g., Other software components of the system):
- GetWindSpeed
- ReadWindSpeed
- SendWindData
- PutWindSpeed

Clock Domain CDKiteControl
- KiteControl
  - ReceiveWindData
  - ReceiveKiteData

Clock Domain CDGetKiteInfo
- GetKiteSpeed
  - SendKiteData
  - Data Ready
  - ReadKiteSpeed

Clock Domain CDGetWindInfo
- GetWindSpeed
  - SendWindData
  - Data Ready
  - ReadWindSpeed
Allows to implement existing languages which only support single threaded-implementation to use multiple threads:

<table>
<thead>
<tr>
<th>SystemJ Statements</th>
<th>Mappings with LibGALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>present S</td>
<td>if (Present(S))</td>
</tr>
<tr>
<td>emit S;</td>
<td>emit(S);</td>
</tr>
<tr>
<td>pause;</td>
<td>pause();</td>
</tr>
</tbody>
</table>
| abort (S)          | strongAbort(S, AbortName); …
|                    | endAbort(S, AbortName); |
Advantages of multi-threaded approach

LibGALS adapts process/thread approach to implement reactions hence reactions can perform when signal dependencies allow – faster in computation time, dependencies resolved dynamically.

LibGALS does not require JVM as SystemJ – smaller code size.

<table>
<thead>
<tr>
<th>Example</th>
<th>Average tick time (µs)</th>
<th>Code Size (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LibGALS</td>
<td>SystemJ</td>
</tr>
<tr>
<td>2CD Freq Relay</td>
<td>27.67</td>
<td>75.23</td>
</tr>
<tr>
<td>2CD KiteController</td>
<td>11.37</td>
<td>27.16</td>
</tr>
<tr>
<td>2CD Async Proto</td>
<td>48.37</td>
<td>16.25</td>
</tr>
<tr>
<td>2CD Data Comp</td>
<td>18.23</td>
<td>26.37</td>
</tr>
<tr>
<td>3CD Data Comp</td>
<td>17.72</td>
<td>39.28</td>
</tr>
<tr>
<td>4CD Data Comp</td>
<td>17.43</td>
<td>56.62</td>
</tr>
</tbody>
</table>
Conclusions

- LibGALS enables designers to describe GALS systems easily.
- GALS systems are collections of concurrent processes.

- LibGALS APIs can be used to abstract out details of communication and synchronization.
  - Less error-prone than using traditional threading libraries.
  - No need to play around with low level constructs.

- LibGALS programs utilize the advantage of multi-processing/multi-core architecture.
  - Better performance!
Future developments

- Dynamic creations of clock domains
  - Clock domains and their reactions can be migrated from one machine to another
  - Possible load sharing or distributed computing to achieve even better performance
  - At this moment, a plug-in system for LibGALS has been developed

- Integrate with other researches to establish a framework to enable system designs
  - Simulate LibGALS programs and SystemC is possible and working under progress
Thank you & Questions?