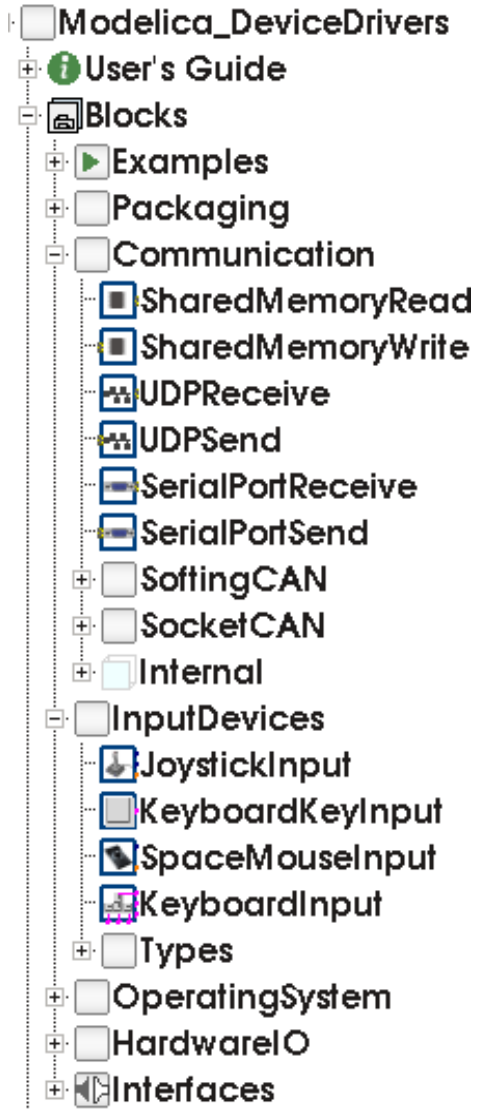
Real-Time Communication and Control

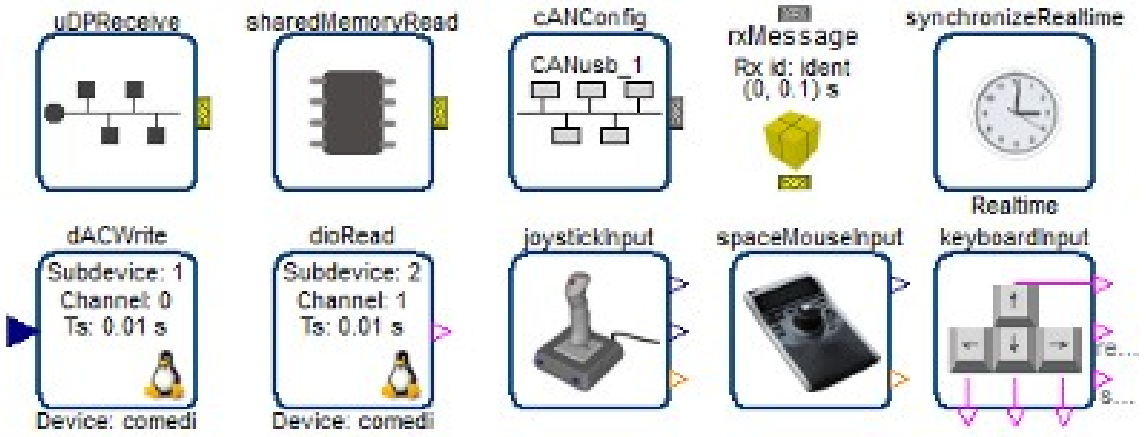
MODELICA_DEVICEDRIVERS Library

Communication & I/O Devices: Modelica_DeviceDrivers Library

Main contributors: Bernhard Thiele, Thomas Beutlich, Volker Waurich, Martin Sjölund, Tobias Bellmann

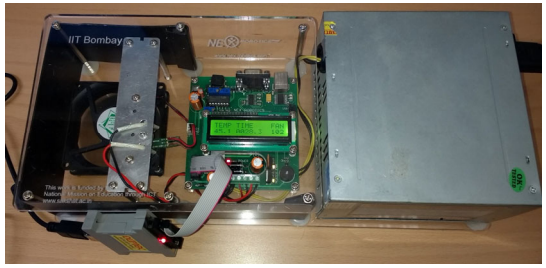


- **Free library** for interfacing hardware drivers
- **Cross-platform** (Windows and Linux)
- UDP, SharedMemory, CAN, Keyboard, Joystick/Gamepad
- DAQ cards for digital and analog IO (only Linux)
- Developed for **interactive real-time** simulations



https://github.com/modelica/Modelica_DeviceDrivers/

Motivation for Modelica Device Drivers Library



Certain Modelica applications need to interact with their environment, *e.g.*,

- **Human-in-the-Loop (HITL) simulators (flight, driving, marine training simulators)**
- **Hardware-in-the-Loop (HIL) simulators**
- **Offline process simulators that need to be coupled to external applications**
- **Embedded control applications**

How to do it?

How to Do It?

- **Established: Code export to third-party environments for real-time applications, as supported by some Modelica tools *Example:* Integration with the MATLAB/Simulink tool chain as supported by Dymola or SimulationX**
- **More recent: Generate FMUs which can be imported into compatible simulator environments**
- **Customization: Self-grown customized tool chains. *Example:***

  N. Worschech and L. Mikelson. **A Toolchain for Real-Time Simulation using the OpenModelica Compiler .** In 9th *Int. Modelica Conference*, 2012.

- **Library approach: Access external devices by utilizing Modelica's external function interface for interfacing to the C API of various device drivers directly from Modelica models**

Library approach: *Modelica_DeviceDrivers* (MDD)

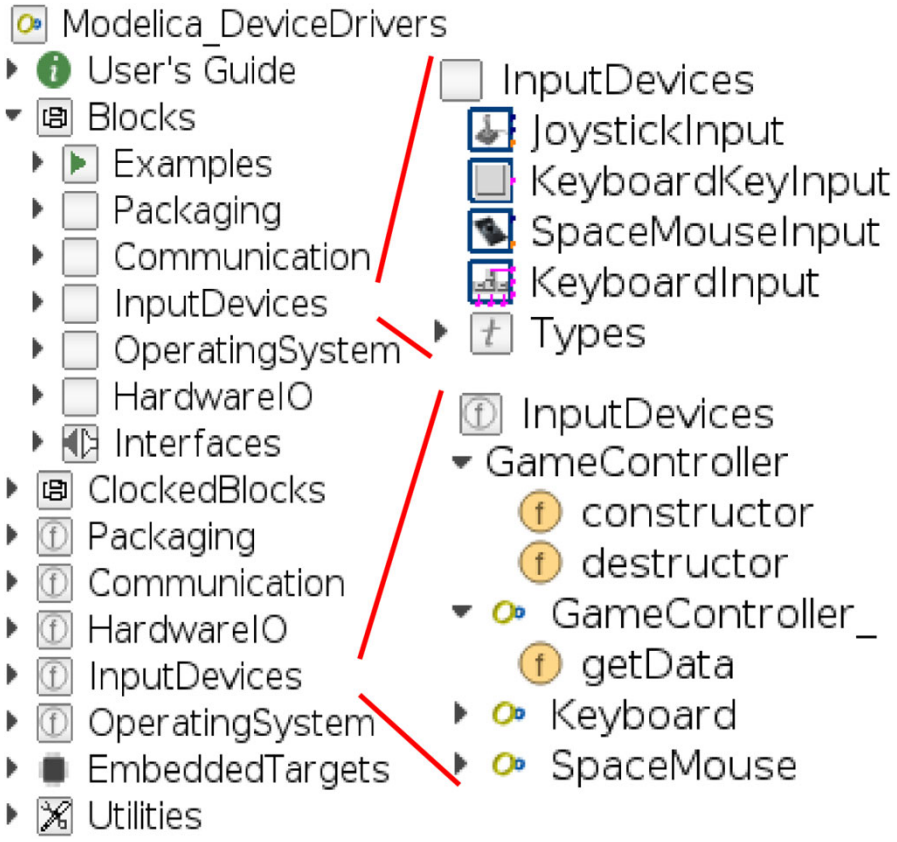
- Open-source Modelica library for interfacing hardware drivers Originates from an DLR internal library named *ExternalDevices*

T. Bellmann. Interactive Simulations and advanced Visualization with Modelica . In 7th *Int. Modelica Conference*, 2009.

- *ExternalDevices* was split into two libraries for separating different concerns and ease further development:
  - *DLR Visualization* library (became a commercial library)
  - *Modelica_DeviceDrivers* (MDD) library (became open-source)
- First public version of MDD v0.9 released in 2012
- Latest release is MDD v1.5.0: Substantial contributions and efforts by several developers since the first concept

https://github.com/modelica/Modelica_DeviceDrivers

Layered Architecture



Block Layer

Blocks	ClockedBlocks
when sample() **then** style for calls to the Function Layer.	*Clocked Synchronous Language Elements* for periodic calls

Function Layer

Modelica (external C) functions

C-Code Layer

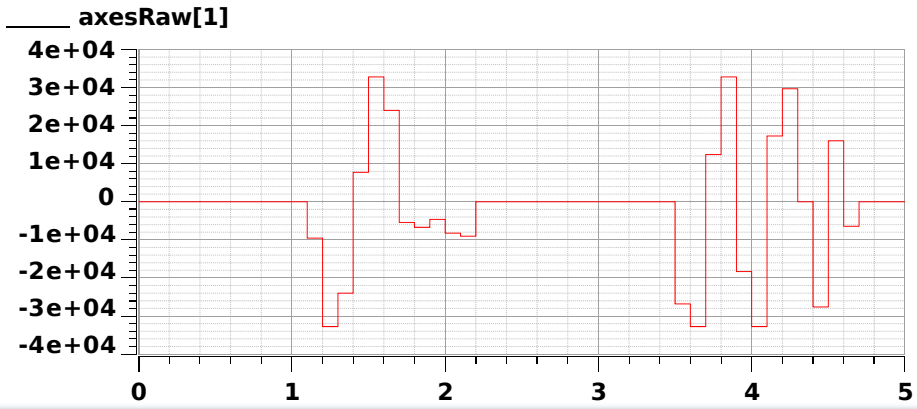
The glue C-code interfaced by the External C-Function Layer.

Windows	Linux	Other

Example: Game controller function layer API

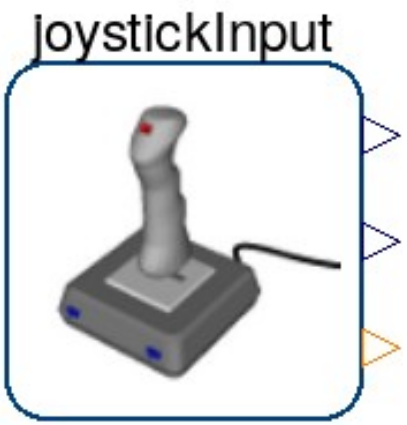
```
model GameControllerExample
  import Modelica_DeviceDrivers.InputDevices.*;
  parameter Integer id=0 "0=first attached game controller";
  GameController gc = GameController(id);
  discrete Real axesRaw[6];
  Integer buttons[32], pOV;
equation
  when sample(0, 0.1) then
    (axesRaw, buttons, pOV) = GameController_.getData(gc);
  end when;
end GameControllerExample;
```

Connect controller and simulate, *e.g.*, in OpenModelica using simulation flag `-rt=1` for real-time synchronisation.

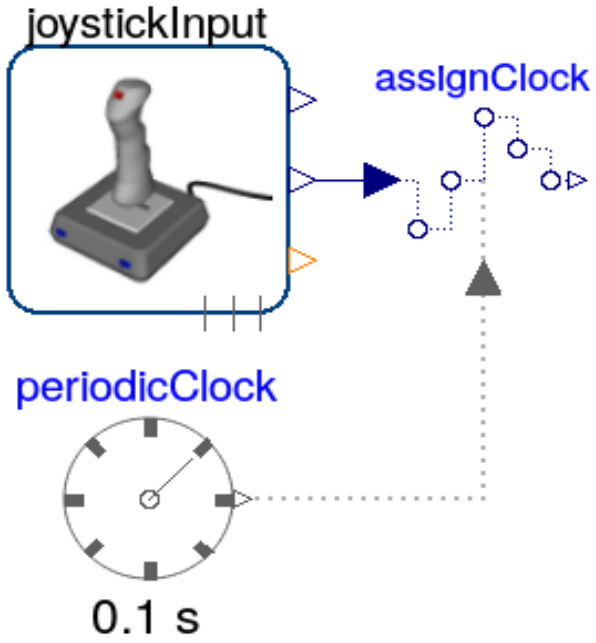


Example: Game controller block layer

Blocks



ClockedBlocks



Sampling period by parameter:

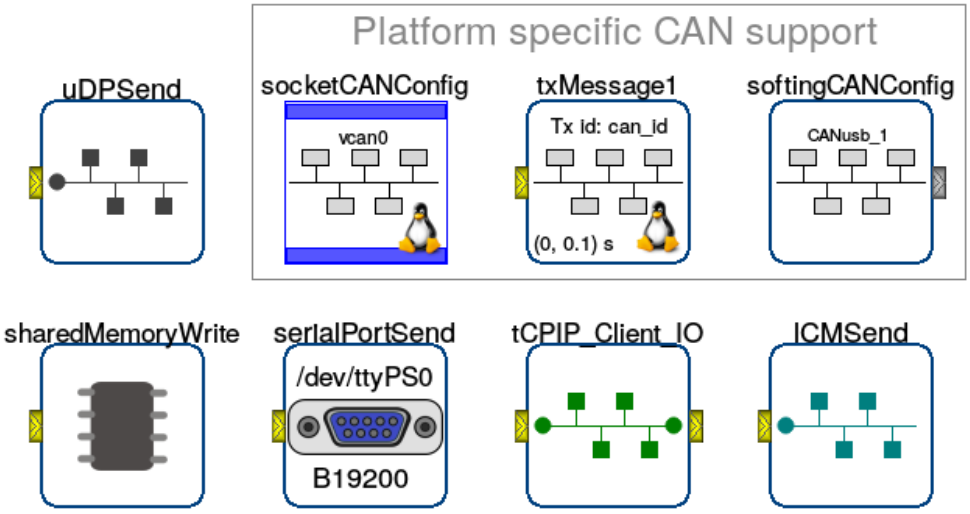
```
when sample (0, sampleTime) then
   (axesRaw, buttons, pOV) =
    GameController_.getData(gc);
end when;
```

***Modelica_Synchronous* compatible:**

```
when Clock() then
   (axesRaw, buttons, pOV) =
    GameController_.getData(gc);
end when;
```

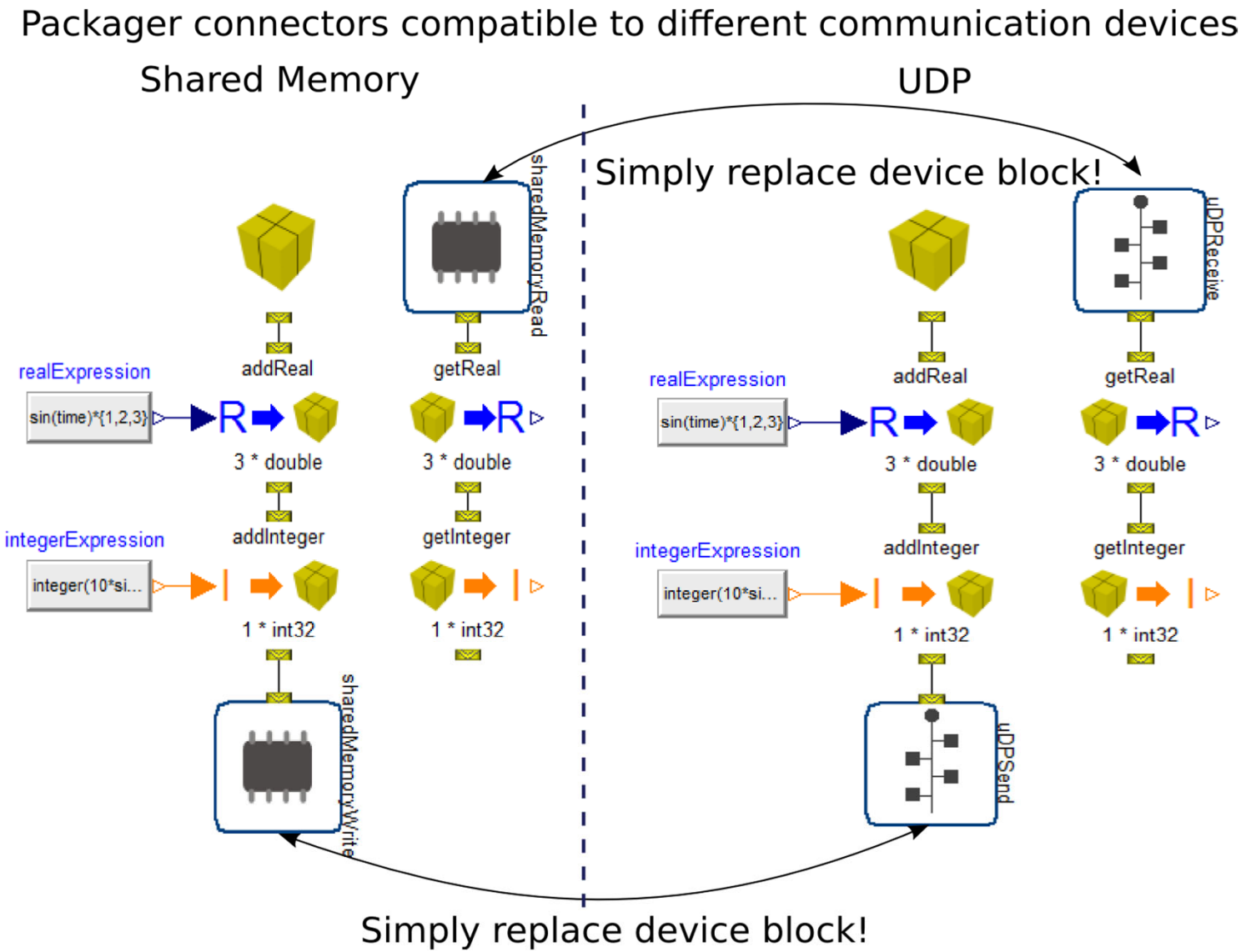
Supported input and communication devices

Blocks.Communication



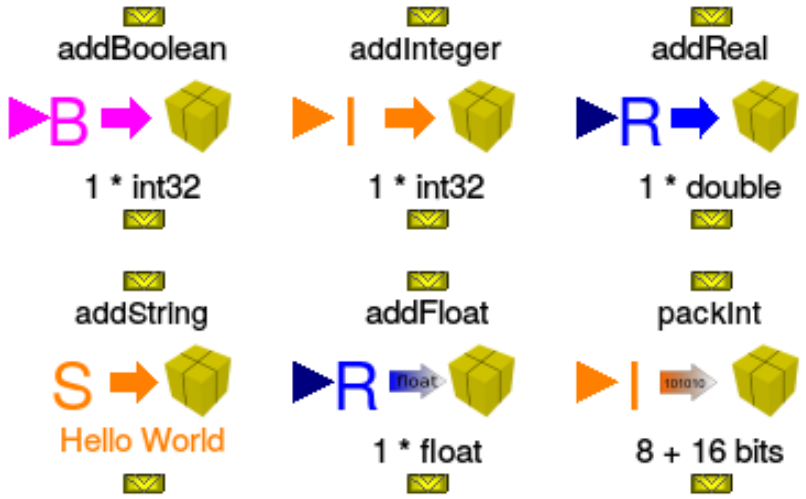
LCM: set of libraries and tools for message passing and data marshalling: https://lcm-proj.github.io/

Packaging concept



SerialPackager blocks and connectors

Blocks.Packaging

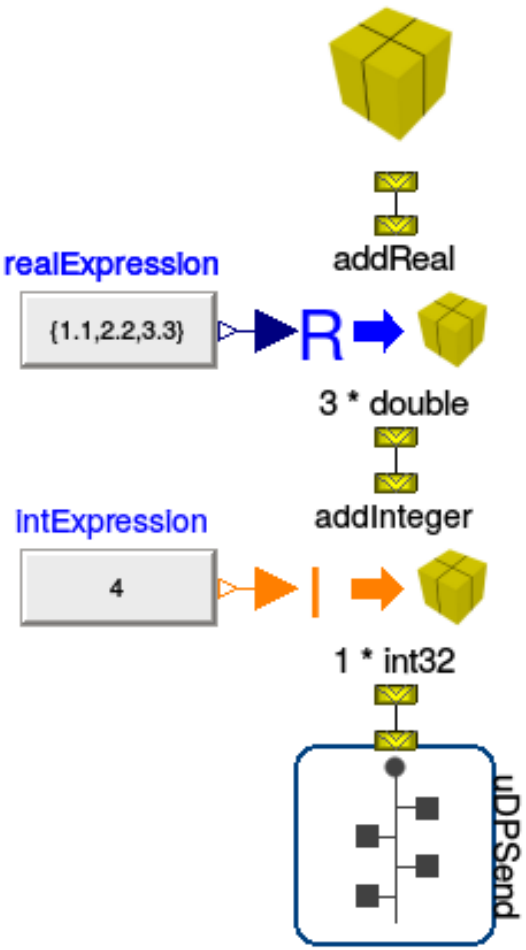


- Symmetrical blocks for retrieving variables
- Byte order can be specified packInt for packing unsigned integer values at the bit level

```
connector PackageIn "Packager input"
    input SerialPackager pkg;
    input Boolean trigger ;
    input Real dummy;
    output Boolean backwardTrigger;
    output Integer userPkgBitSize;
    output Integer autoPkgBitSize;
end PackageIn;
```

- Symmetrically, PackageOut has reversed causalities
- pkg is an external object pointing to a C byte array (next slide)

Serial packager external C byte array structure

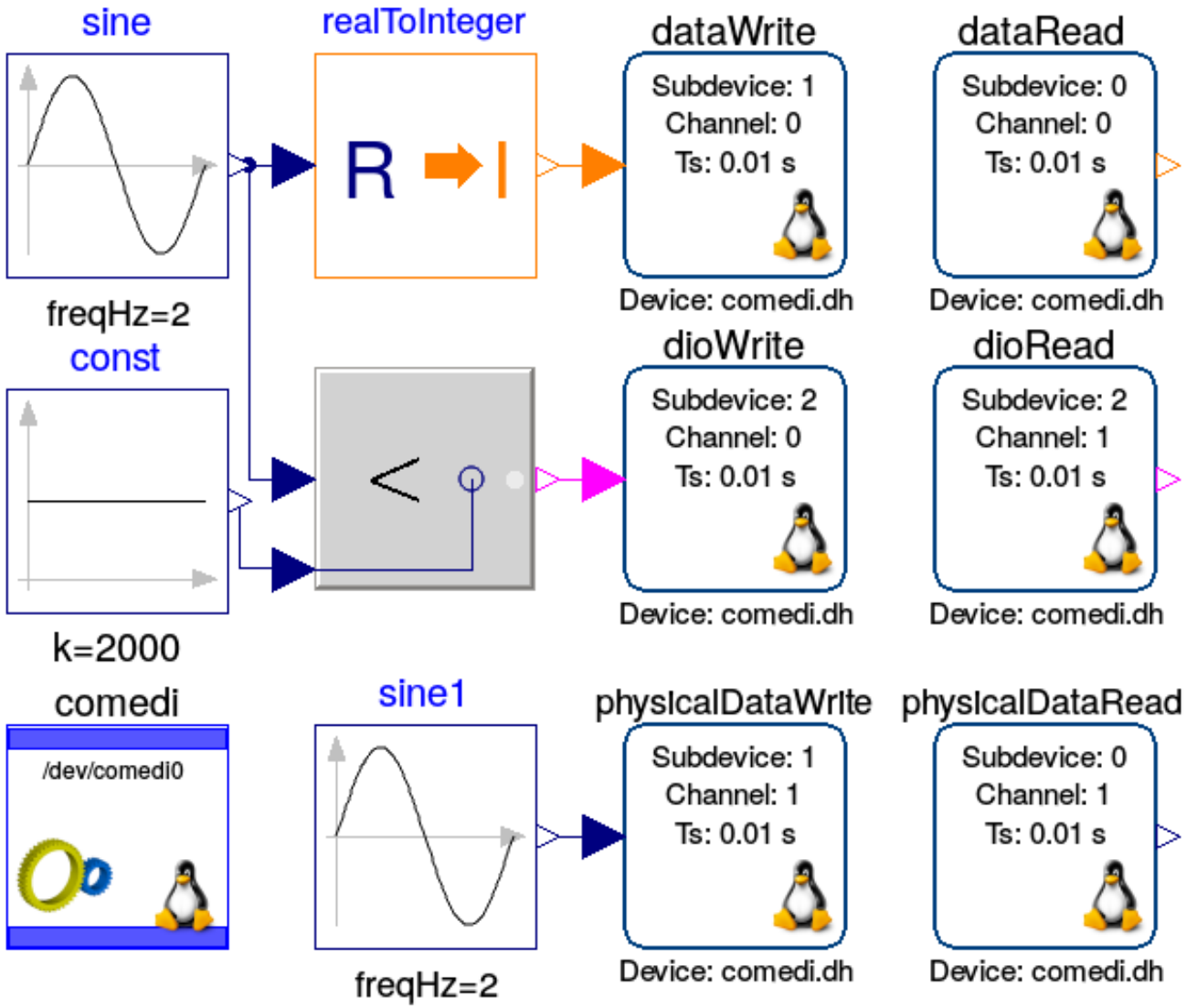


- pkg objects are propagated through connections
- pkg points to a C byte array in which C variables with respectively indicated types are simply successively appended in a data-flow prescribed order

Conceptually (using magic numbers for array offsets):

```
double v1[3] = { 1.1, 2.2, 3.3 };
int v2 = 4;
unsigned char * data =
  (unsigned char*)calloc(28, sizeof(unsigned char));
memcpy(&data[0], &v1[0], sizeof (v1));
memcpy(&data[2 4], &v2, sizeof (v2));
```

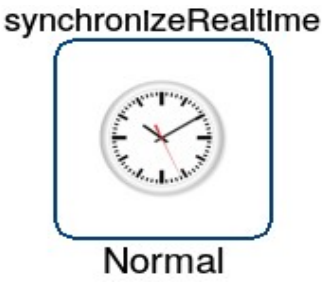
HardwareIO: Support for the Linux control and measurement device interface ”COMEDI”



Real-time synchronization

- MDD models typically need real-time synchronization
- For interactive simulations *soft* r.-t. is often sufficient

Blocks.OperatingSystem



- MDD contains a block for (soft) real-time synchronization Modelica tools may provide better (tool-specific) options!
- Standard Windows/Linux desktops ≈ soft r.-t., but low-latency kernels can push limits

Modelica's External Function Interface

```
function getKey
  input Integerkey Code "Key code";
  output Integer keyState "Key state" ;
external "C" MDD_keyboardGetKey ( keyCode , keyState )
  annotation(
        Include = "#include \"MDDKeyboard.h\"",
        Library  = {"X11", "User32" } );
  annotation(__ModelicaAssociation_Impure = true);
end getKey;
```

- Include header file MDDKeyboard.h ⟹ use # if , # elif , etc. in header file for platform differentiation Link with libraries X11 and User32
- Declare function impure ⟹ don't optimize call away!

X11↔Linux, User32↔Windows, but this cannot be stated

Workaround: Link with all libraries and provide dummy libraries for libraries missing for a platform

Serial packager

External Object Aliasing

- External objects are propagated through connections
- Hence, aliasing equations are generated (in.pkg = out.pkg)
- Hence, objects are not explicitly created by calling an external object constructor

not allowed in Modelica v3.3!

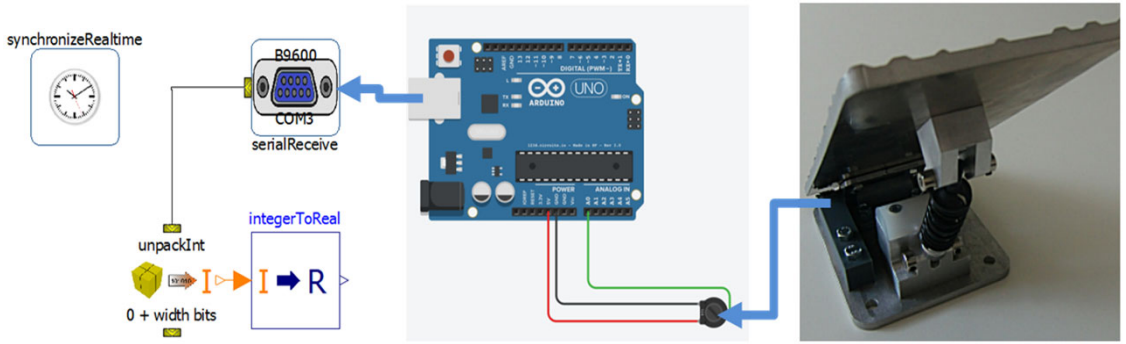
Automatic Buffer Size

- Ext. obj. constructor needs bufferSize of C byte array
- Setting bufferSize manually is possible, but bufferSize can also be deduced automatically by solving a system of initial equation

not supported by all Modelica tools and a specification corner case!

DLR and TUD Demonstrators

- DLR Institute of System Dynamics and Control: DLR Robotic Motion Simulator & other demonstrators
- Force-feedback steering wheels connected via CAN blocks
- Consumer based input devices such as pedals via JoystickInput block



TUD Chair of Construction Machinery: Customized control devices with Arduino connected via serial port blocks to real-time simulation. Wireless by using Bluetooth module with Serial Port Profile (SPP).

Linköping University, Embedded Control Demonstrator SBHS (Single Board Heating System)

Single board heating system (from IIT Bombay)

- Use for teaching basic control theory
- Usually controlled by serial port (set fan value, read temperature, etc)
- OpenModelica can generate code targeting the ATmega16 on the board (AVR-ISP programmer in the lower left).

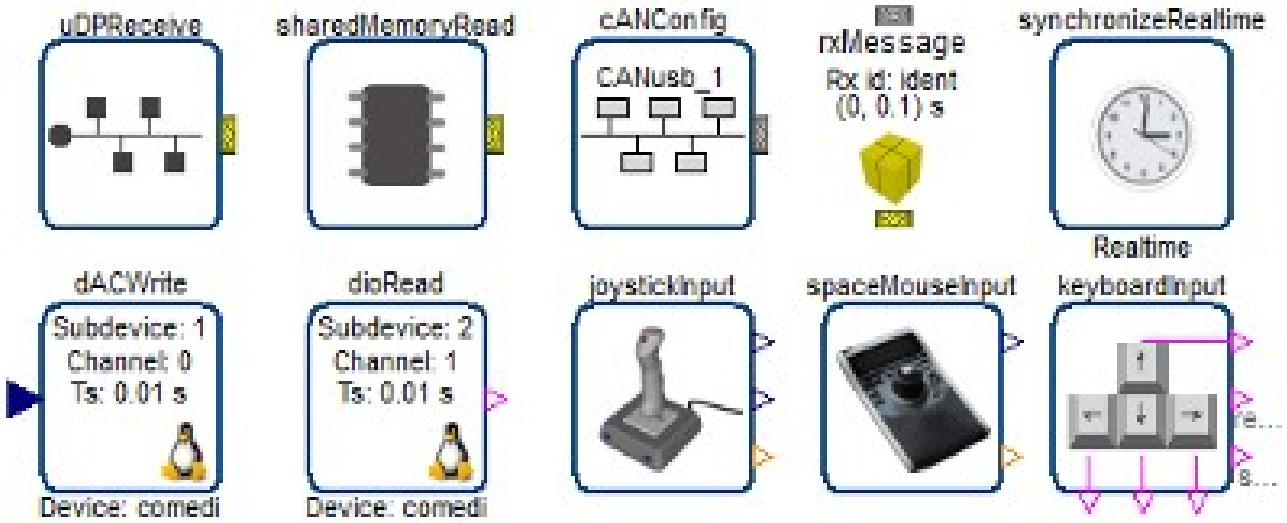
Program size is 4090 bytes including LCD driver and PID-controller (out of 16 kB flash memory available).

Movie Demo, see next page!

SBHS (Single Board Heating System)

Wrap-up: *Modelica_DeviceDrivers*

- **Free library for interfacing hardware drivers Cross-platform (Windows and Linux)**
- **UDP, SharedMemory, CAN, Keyboard, Joystick/Gamepad, … DAQ cards for digital and analog IO (only Linux)**
- **Developed for interactive real-time simulations**
- **Works in Dymola, SimulationX, and (for the most part) OpenModelica**

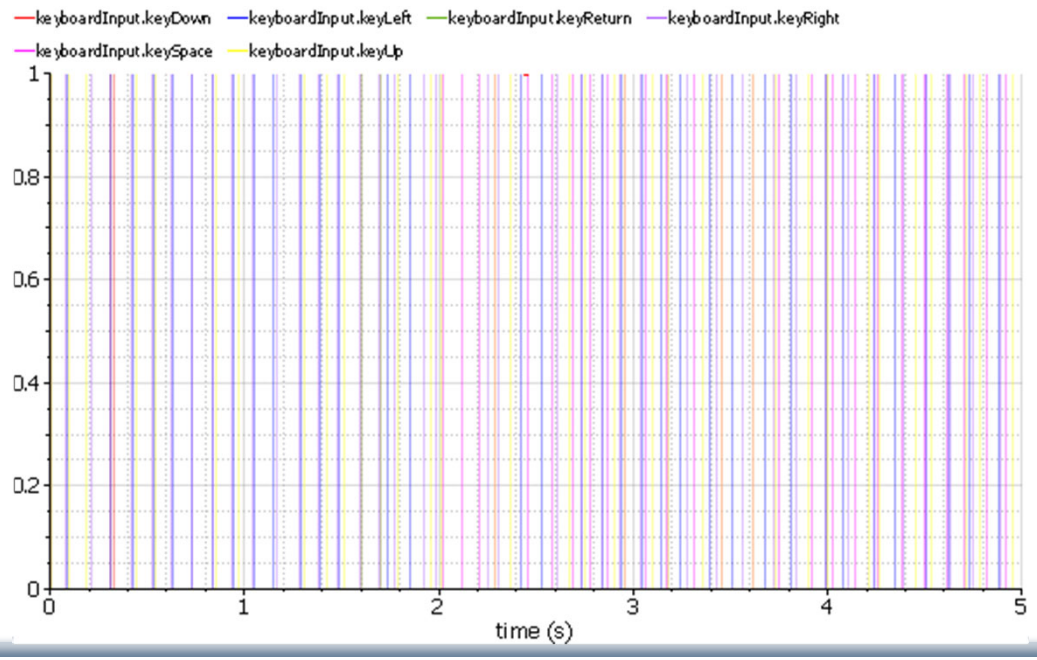


Outlook

- Many conceivable future extensions due to myriad number of (hardware) devices and communication protocols
- Frequently requested: Revised data serialization, possibly using LCM, MessagePack, or similar
- Internet of Things (IoT) related extension, *e.g.*, support mqtt protocol
- Support of embedded systems beyond the current prototypical work
- Hopefully, further Modelica standard improvements for facilitating the development of cross-platform, external C code dependent libraries
- Thanks to all contributors and organisations that supported MDD
- Pull requests welcome …

Exercise – plot keyboard input

- Load Modelica_DeviceDrivers in OMEdit
  File->System Libraries->Modelica_DeviceDrivers
- Open Model: Modelica_DeviceDrivers.Blocks.Examples.TestInputKeyboard
- Simulate the model TestInputKeyboard and during simulation press keys: space, down, up, left, right in any order you want and as many times as you want
- Plot keyboardInput.*



Resources

https://github.com/modelica/Modelica_DeviceDrivers

External Objects in Records

- SocketCan and Comedi blocks use a Modelica record for specifying general hardware settings
- Approach: Instantiate record once and pass it to blocks using the device

```
record ComediConfig
  parameter String deviceName = "/dev/comedi0" "Name of Comedi device";
 final parameter Comedi dh = Comedi(deviceName) "Handle to Comedi device";
end record;
```

Can be interpreted as the record returning the object and assigning it to another external object (forbidden in Modelica v3.3)