The 11th International workshop on OpenCL and SYCL

IWOCL & SYCLCON 2023

Accelerating Simulink/Matlab projects with SYCL

Uwe Dolinsky, Codeplay Software Limited

April 18-20, 2023 | University of Cambridge, UK

iwocl.org

Company

Leaders in enabling high-performance software solutions for new AI processing systems

Enabling the toughest processors with tools and middleware based on open standards

Established 2002 in Scotland, now with ~90 employees. Acquired by Intel in 2022.

Collaborations

SYNOPSYS®
BROADCOM
CEVA
Imagination
RENESAS
KUC Kyoto Microcomputer Co., Ltd.
NSI-TEXE
BERKELEY LAB
OAK RIDGE National Laboratory
Argonne NATIONAL LABORATORY

And many more!

Supported Solutions

oneAPI

An open, cross-industry, SYCL based, unified, multiarchitecture, multi-vendor programming model that delivers a common developer experience across accelerator architectures

ComputeCpp™

C++ platform via the SYCL™ open standard, enabling vision & machine learning e.g. TensorFlow™

ComputeAorta™

The heart of Codeplay's compute technology enabling OpenCL™, SPIR-V™, HSA™ and Vulkan™

Markets

High Performance Compute (HPC)
Automotive ADAS, IoT, Cloud Compute
Smartphones & Tablets
Medical & Industrial

Technologies: Artificial Intelligence
Vision Processing
Machine Learning
Big Data Compute

Why accelerating Simulink/Matlab with SYCL

- Many Automotive/Avionics/Defence/Embedded software projects are entirely written in Simulink/Matlab
  - Companies have large Simulink/Matlab legacy code bases
  - Engineers have mainly expertise in Matlab/Simulink using specialised commercial Matlab Simulink tool boxes
  - Targeting Safety Critical
- Taking advantage of open source tool chains and acceleration ecosystems
  - Ability to accelerate on more types of platforms via SYCL
  - Use different open libraries as backends (Eigen, Armadillo etc)
  - Ability to apply more and different verification tools to Simulink/Matlab projects
- Matlab/Simulink are widely taught in engineering and sciences
- Many research projects/codes in engineering are based on Matlab (or Matlab-like software like Octave, COMSOL, SciLab, …)

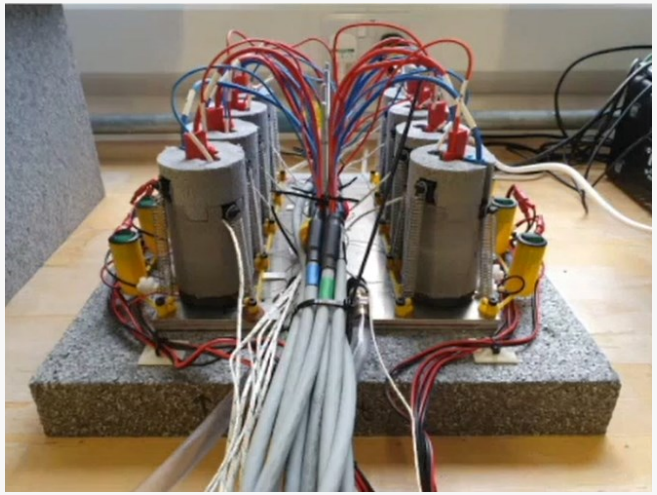
Context 1/2: Building novel High-Performance Hybrid Batteries for Electric Vehicles

IUK-funded project (WIZer Batteries, grant no. 104427)

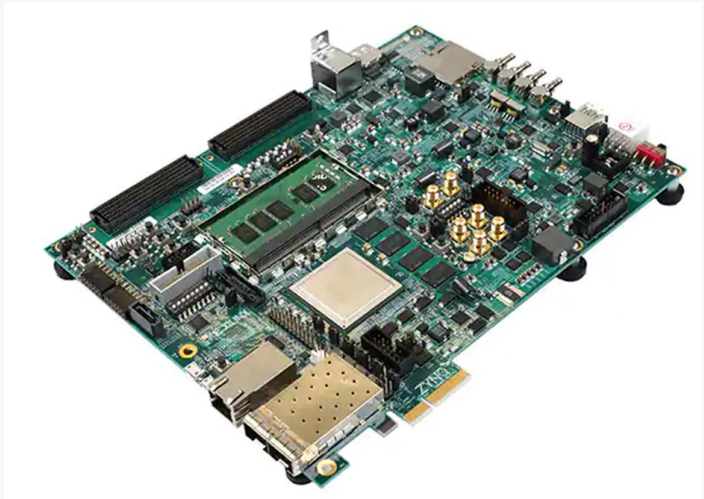
Collaboration led by Williams Advanced Engineering.

Codeplay's role: Accelerating Battery Models via SYCL.

Project partners



Experimental Battery Test rig at Imperial. (Image from www.imperial.ac.uk)



Embedded MPSoC platform running the BMS on the Battery. (Image from https://www.xilinx.com/products/boards-and-kits/zcu106.html)

Context 2/2: SYCL Acceleration flow for Simulink/Matlab

Battery Model Architecture maintained by WAE in MatLab/Simulink format

Normal Matlab flow

Matlab/Simulink Coder

MathWorks

C/C++

SYCL flow

WIZer Coder

WIZer

C/C++ with SYCL bindings

ComputeCpp

SYCL support (CPU, FPGA)

SYCL

Open Accelerated SYCL Libraries

BMS Software Architecture

WIZer

- **Simulink/Matlab translation to SYCL/C++**
- **SYCL code acceleration on CPU/FPGA**

Integrating open source libraries (Eigen) and tools to read Matlab/Simulink file formats.

What is Simulink/Matlab?

- Enables building/testing complex programs/models using a graphical environment
- Models are graphs of blocks – blocks can be simple arithmetic operations up to complex subsystems, dealing with data/storage/constants, StateFlow, Control flow, Math Operations
  - Frequently used block types: `From`, `Goto`, `Inport`, `Outport`, `Reference`, `SubSystem`, `Selector`, `Concatenate`, `Terminator`, `Switch`, `RelationalOperator`, `MinMax`, `UnitDelay`, `Product`, `Sum`
- Each block has inports, outports and block parameters
- Matlab/Simulink Interactions
  - Blocks can have Matlab code attached to them:
    - expressions or whole programs with multiple functions referencing external functions
  - Matlab configures workspace and can call/query Simulink model

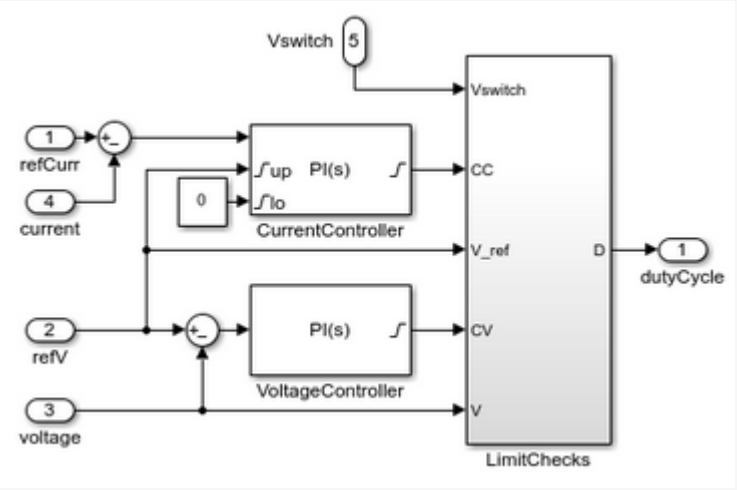


Image from: https://www.mathworks.com/help/sps/ug/smartphone-charging.html

A typical Simulink/Matlab project contains

- One or several Simulink solution files (*.slx, *mdl)
  - Simulink libraries
- Matlab source files (*.m, workspace setup, free functions, embedded/anonymous functions)
- Matlab data files
  - Matlab data files (*.dat, workspace data, simulation data, variables)
  - Data dictionaries (*.sld)

Our approach (technical and user targeted)

- Converting Simulink/Matlab solutions into C++ code (**the model step**) that uses an API that can be accelerated via SYCL.
- Taking advantage of **open source projects** to
  - Import Matlab data files (*.mat)
  - Import Simulink Solution files (*.mdl, *.slx)
  - Provide efficient Vector/Matrix/Math operations (via Eigen)
- No dependencies on MATLAB® installation
- Non-disruptive: Enabling engineers to continue to develop in Simulink/Matlab
- Option to run entire model step or individual blocks as SYCL kernels
- Only functionality required by BMS use case was implemented

Other/Related MATLAB projects

Integrating Intel® Data Analytics Acceleration Library with Matlab

https://www.intel.com/content/www/us/en/developer/articles/technical/using-intel-data-analytics-acceleration-library-on-matlab.html

CoCoSim: Open-Source verification tool for Simulink Models

- Requires MATLAB® Installation

https://github.com/NASA-SW-VnV/CoCoSim

m2cpp: Open-source tools to convert process Matlab/Simulink files

- converts Matlab files only (research project)

https://github.com/emc2norway/m2cpp

Challenges of our Integration

- Integrating output from various open tools/libraries to process
  - Simulink solutions files
  - Data files (and dictionaries)
  - Matlab files (workspace setup files, Matlab functions (free, embedded, anonymous)
- Support different block types
  - Models consist of the main model, sub models, model references, subsystems (virtual or atomic) and are connected by ports
- Generating model step which evaluates the entire model
  - Determine execution order of blocks
  - Requires block scheduling to flatten model graph for execution
- Outputting the model and data as C++ code and integrating with backend (Eigen)
- Targeting SYCL
- Provide flexibility/configurability to enable performance tuning

Open-source projects used to help translate Simulink and Matlab files

- Simulink files read by the ConQAT project
  https://github.com/vimaier/conqat/

- Matlab data files read by **MAT File Library**
  https://github.com/HebiRobotics/MFL

Example Simulink model translated into C++

```
#include "matlab_includes.h"
struct {
  double Ian1;
  double In2;
  double Ian2;
  double In1;
} rootInports;
struct {
  double Out1;
  double Out;
  double Ot3;
  double Out2;
  double Out3;
  double Ot1;
} rootOutports;

template <class Name, class T> void ModelStep(T &cgh, StepState *__state) {

  // new_mod/bbbbbbld1 [Sum, 2:1]
  decltype(auto) Val0 = cp_subtract(-2, rootInports.Ian2);
  CP_LOG_PORT("new_mod/bbbbbbld1 [Sum, 2:1]", 1, Val0)

  // new_mod/bbbbbbld [Sum, 2:1]
  decltype(auto) Val1 = cp_subtract(-2, rootInports.Ian1);
  CP_LOG_PORT("new_mod/bbbbbbld [Sum, 2:1]", 1, Val1)

  // new_mod/ppppppppt [Product, 2:1]
  decltype(auto) Val2 = cp_dot_star(Val1, rootInports.In2);
  CP_LOG_PORT("new_mod/ppppppppt [Product, 2:1]", 1, Val2)

  // new_mod/ppppppppt2 [Product, 2:1]
  decltype(auto) Val3 = cp_dot_star(Val0, rootInports.In1);
  CP_LOG_PORT("new_mod/ppppppppt2 [Product, 2:1]", 1, Val3)

  // new_mod/Addddddddddd1 [Sum, 2:1]
  decltype(auto) Val4 = cp_sum(Val0, Val3);
  CP_LOG_PORT("new_mod/Addddddddddd1 [Sum, 2:1]", 1, Val4)

  // new_mod/Addddddddddd [Sum, 2:1]
  decltype(auto) Val5 = cp_sum(Val1, Val2);
  CP_LOG_PORT("new_mod/Addddddddddd [Sum, 2:1]", 1, Val5)

  // new_mod/aaaaaa2 [Sum, 2:1]
  decltype(auto) Val6 = cp_subtract(Val4, Val3);
  CP_LOG_PORT("new_mod/aaaaaa2 [Sum, 2:1]", 1, Val6)

  // new_mod/aaaaaa1 [Sum, 2:1]
  decltype(auto) Val7 = cp_subtract(Val5, Val2);
  CP_LOG_PORT("new_mod/aaaaaa1 [Sum, 2:1]", 1, Val7)

  // new_mod/ppppppppt3 [Product, 2:1]
  decltype(auto) Val8 = cp_dot_div(Val6, 32767);
  CP_LOG_PORT("new_mod/ppppppppt3 [Product, 2:1]", 1, Val8)

  // new_mod/ppppppppt1 [Product, 2:1]
  decltype(auto) Val9 = cp_dot_div(Val7, -32768);
  CP_LOG_PORT("new_mod/ppppppppt1 [Product, 2:1]", 1, Val9)

  cp_assign(rootOutports.Out3, Val3);
  cp_assign(rootOutports.Out2, Val4);
  cp_assign(rootOutports.Out1, Val5);
  cp_assign(rootOutports.Ot1, Val8);
  cp_assign(rootOutports.Out, Val2);
  cp_assign(rootOutports.Ot3, Val9);
}
```

API header

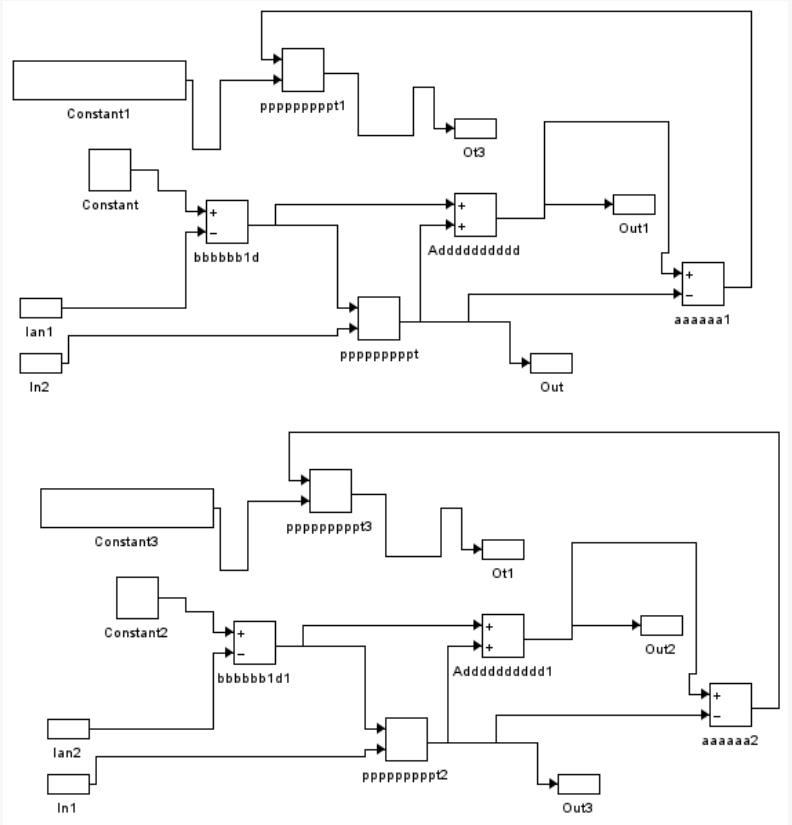
model inports

model outports

model step function

Writing model outports

Example model (25 blocks) from
https://sourceforge.net/projects/sim2c/



Model architecture rendered by open-source ConQAT library
https://github.com/vimaier/conqat/

Simulink model translated into C++ - verbose block information

```
template <class Name, class T> void ModelStep(T &cgh, StepState *__state) {

  // new_mod/bbbbbb1d1 [Sum, 2:1]
  //   BlockMirror:  off
  //   FontWeight:  normal
  //   Name:  bbbbbb1d1
  //   CollapseDim:  1
  //   FontAngle:  normal
  //   ForegroundColor:  black
  //   BlockRotation:  0
  //   Inputs:  +-
  //   InputSameDT:  off
  //   NamePlacement:  normal
  //   AccumDataTypeStr:  Inherit: Inherit via internal rule
  //   OutMax:  []
  //   Position:  [175, 487, 205, 518]
  //   OutDataTypeStr:  Inherit: Inherit via internal rule
  //   Ports:  [2, 1]
  //   SaturateOnIntegerOverflow:  off
  //   IconShape:  rectangular
  //   LockScale:  off
  //   RndMeth:  Floor
  //   SampleTime:  -1
  //   ShowName:  on
  //   DropShadow:  off
  //   CollapseMode:  All dimensions
  //   SID:  49
  //   OutMin:  []
  //   FontSize:  10
  //   BlockType:  Sum
  //   FontName:  Helvetica
  //   BackgroundColor:  white
  decltype(auto) Val0 = cp_subtract(-2, rootInports.Ian2);
  CP_LOG_PORT("new_mod/bbbbbb1d1 [Sum, 2:1]", 1, Val0)

  // new_mod/bbbbbb1d [Sum, 2:1]
  //   BlockMirror:  off
```

Example model from https://sourceforge.net/projects/sim2c/

Outputting detailed parameters of each block as C++ comments improves model introspection and debugging.

Integration of Eigen for math operations

```
#include "matlab_includes.h"
struct {
  double Ian1;
  double In2;
```

Eigen is imported through API header

```
// new_mod/bbbbbb1d1 [Sum, 2:1]
decltype(auto) Val0 = cp_subtract(-2, rootInports.Ian2);
CP_LOG_PORT("new_mod/bbbbbb1d1 [Sum, 2:1]", 1, Val0)
```

Overloads with Eigen operations selected if block argument types are types from Eigen Library

```
template <int rows, int cols, class Type>
struct cp_matrix
#ifdef CP_USE_EIGEN
  typedef typename std::conditional<
      (rows > AGGREGATE_THRESHOLD || cols > AGGREGATE_THRESHOLD), AggregateType,
      Eigen::Matrix<Type, rows, cols>>::type matr_type;
  matr_type m_impl___;
#else
  // other backend
#endif
  // declarations
};
```

Defined inside `matlab_includes.h`

`cp_matrix` is a core data type encapsulating Eigen::Matrix

SYCL Integration

- Targeting Simulink For Each Subsystems: https://www.mathworks.com/help/simulink/slref/foreachsubsystem.html

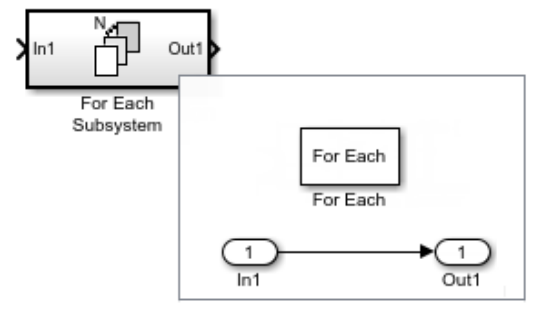


Image taken from above URL
© The MathWorks, Inc.

  - Subsystem containing a **ForEach** block with control parameters (Dimension)
  - Essentially turns a subsystem into an array of subsystems (duplicating state)
    - A subsystem is a set of blocks
  - Apply SYCL's `parallel_for` to the array to evaluate each subsystem
- Targeting SYCL (dealing with Matlab "persistent" and "global" data inside functions – requires transforming code to hoist these variables out of the function).

SYCL Integration

```
struct {
  template <class T, class T2> auto operator()(T, T2) {
     // execute the model step of the ForEach Subsystem

     // return model outputs
  }
} foreach_var2[12]; //static storage

template <class Name, class T> void ModelStep(T &cgh, StepState *__state) {


  decltype(auto) fe_ret3 = process_mul_fe(cp_foreach<ForEachNum1>(__state,
                                          foreach_var2, arg1, arg2));
  decltype(auto) Val9  = cp_get_element<1>(fe_ret3);
  decltype(auto) Val10 = cp_get_element<2>(fe_ret3);
  // more code
}
```

Concatenating results

Kernel launch

Kernel name

SYCL state (queue)

Array of SubSystems

```
g++ file.cpp -o cpp_app
clang++ -fsycl -DUSE_SYCL file.cpp -o sycl_app
```

Building as C++ application

Compiling for SYCL

Each ForEach system generates an array of subsystem structs/functors.

In SYCL mode Each subsystem in the array is evaluated in parallel.

Configuring SYCL integration

- Running all For Each systems as SYCL kernels may not improve performance.
- Generated model file provides option to selectively compile functions as normal CPU functions instead.

```
#include "matlab_includes.h"

  // To build a kernel in SYCL mode as C++ function just #define its name for example on the command line, e.g. -DForEachNum3

  // Kernel names:
  //    ForEachNum1
  //    ForEachNum2
  //    ForEachNum3
  //    ForEachNum4
  //    ForEachNum5
```

Names of the kernels generated for each For Each system

An autotuner could determine which kernels need to be SYCL kernels or CPU functions to maximise performance.

Limitations of current Simulink support

- Only a subset of Simulink Blocks are supported
- On supported blocks only a subset of block parameters are supported
- Supported Simulink/Matlab file formats
  - Simulink solution files need to be in Matlab 2020 format (or older)
    - Open source project could be extended to provide more recent file formats
    - Many Simulink projects can be converted – requires Matlab installation
  - Matlab data files need to be in format version 5 (OpenSource Octave can be used to convert data files)
- No GUI or Simulink commands supported in Matlab code
- Backend limitations: Eigen does not support all of Simulink/Matlab operations (interpolations, etc), using some unsupported Eigen modules
- Currently using synchronous execution to evaluate subsystems
- Task parallelisation opportunities not exploited yet

Future ideas

- Targeting new (updated) use-case
- Auto-tuning to decide which kernel runs where (C++, CPU, GPU,) and with what parameters
- Asynchronous execution of SYCL kernels
- Eigen SYCL backend integration
- Generate different block schedules for performance
- Task parallelisation
- Improve Buffer creation, support USM
- Targeting SYCL to other block types
- Targeting SYCL **for Safety Critical Systems**
- Potentially open sourcing
- Writing paper with performance numbers on open-source use case

Wrap

- Presented open-source- based tool flow to translate Simulink/Matlab into C++ - no dependency on Matlab tools
- Enables acceleration via SYCL (For Each systems)
- Generates static model structure – deterministic model evaluation (at least in C++ without SYCL)
  - Required for automotive use case
- Successfully applied to industrial use case (Battery Models running on embedded platform)
  - Non-disruptive – enables Simulink engineers to take advantage of SYCL
- Generated C++/SYCL code provides options to:
  - evaluate the entire model step or just (For Each blocks) as SYCL kernel
  - selectively enable SYCL per individual block

Acknowledgements and disclaimer

- This work was partly supported by the Innovate UK WIZer project (grant no. 104427)

Notices & Disclaimers

Technologies may require enabled hardware, software or service activation.
No product or component can be absolutely secure.
Your costs and results may vary.

codeplay

Enable AI & HPC to be Open, Safe and Accessible to All

Thank You!

@codeplaysoft

info@codeplay.com

codeplay.com