Software Synthesis for Control System Algorithms in Industrial Applications

Emmanuel Roy – The MathWorks
WorkShop on Software Synthesis
Friday, Oct. 16th, 2009, Grenoble, France
Model-Based Software Synthesis Overview

Plant or Environment

Controller or Application

Embedded System
Model-Based Software Synthesis Overview

- High level of Abstraction:
  - Dynamic typing
  - Dynamic sizing
  - Type and rate propagation
  - Concise math representation
  - ...

- Hierarchical Algorithm design
  - Complexity decomposition
  - Reuse

- "Simulable" (executable specification)
- Multi-Domain Modelling
- Early Verification&Validation
- Powerful visualization possibilities
- ...

Embedded System
Model-Based Software Synthesis Overview

Simulink

Stateflow

Embedded MATLAB

Simscape

Multiple analyses and optimizations

Intermediate Representation (IR)

Analysis/Transform

Analysis/Transform

Target Language Backend

C

C++

HDL

Multiple domains

Multiple Targets
Model-Based Software Synthesis Overview

Plant or Environment

Controller or Application

Embedded System
Model-Based Software Synthesis Key Benefits

- Separate algorithm from implementation
- Parameterizeable implementations
- Target independence
- Flexibility and reusability
- Multi (targeted) language support
Model-Based Software Synthesis Key Benefits

- The model expresses the algorithm concisely while hiding operation’s implementation details:

```c
void fahrenheit2celsius(void) {
    int32_T i;
    for (i = 0; i < 10; i++) {
        Celsius[i] = (Fahrenheit[i] - 32.0) * 5.5555555555555558E-001;
    }
}
```
Model-Based Software Synthesis Key Benefits

- The model-based algorithm is independent of the integration platform:

```c
void fahrenheit2celsius (void) {
    Celsius = (Fahrenheit - 32.0) * 5.5555555555555558E-001;
}
```

```c
void fahrenheit2celsius (const real_T Fahrenheit, real_T* Celsius) {
    (*Celsius) = (Fahrenheit - 32.0) * 5.5555555555555555E-001;
}
```

```c
void fahrenheit2celsius(void) {
    real_T rtb_Fahrenheit;
    Rte_Read_RPort_Fahrenheit(&rtb_Fahrenheit);
    Rte_Write_PPort_Celsius((rtb_Fahrenheit - 32.0)*5.5555555555555555E-001);
}
```
Model-Based Software Synthesis Key Benefits

- Ensure high flexibility and reusability

```c
void caller(void) {
    ...  
    fahrenheit2celsius(&input1, &rtb_tmp1);
    fahrenheit2celsius(&input2, &rtb_tmp2);
    output = rtb_tmp1 - rtb_tmp2;
}
```
Model-Based Software Synthesis Key Benefits

- The model-based algorithm is independent of the specific target optimizations

```c
int32_T tmp;
tmp = (int32_T)input1 * (int32_T)input2;
if (tmp > 32767L) {
    output = MAX_int16_T;
} else if (tmp <= -32768L) {
    output = MIN_int16_T;
} else {
    output = (int16_T)tmp;
}
```

Portable ANSI-C (with overflow protection)

Compagny wide math operation

```c
compagny_XXX_optimized_mul(&output, input1, input2);
```

Target specific optimized math operation

```c
output = c28x_mul_s16_s16_s16_sat(input1, input2);
```
Model-Based Software Synthesis Key Benefits

- Automatic mapping of functional rates into software tasks (RM based scheduler)

```c
void task_fastest(void) {
    ...  /* RateTransition: '<Root>/RateTransition' */
    if ((rtM->Timing.RateInteraction.TID0_1 == 1)) {
        rtB.RateTransition = rtDWork.RateTransition_Buffer0;
    }  
    ... 

void task_slowest(void) {
    ... 
}
```
Model-Based Software Synthesis Key Benefits

- Multi-Language support

Verilog

```verilog
assign mul_temp = input_ravd * b_1_under_gainparam;
assign b_1_out1 = ((2*(mul_temp[31])), mul_temp[31:0])
always @(posedge clk or posedge reset)
begin: BodyDelay2_process
  if (reset == 1'b1) begin
    BodyDelay2_out1 <= 0;
  end
  clk begin
    if (mb_1_10_0 == 1'b1) begin
      BodyDelay2_out1 <= Input_signed;
    end
  end
end // BodyDelay2_process
```

VHDL

```vhdl
mul_temp <= input_signed * b_1_under_gainparam;
b_1_out1 <= resize(shift_right(mul_temp(31)) & mul_temp(31 DOWNTO 0))
```

C

```c
void f2c_step(void) {...}
```

C++

```cpp
class f2c {
public:
    double compute(const double input);
    f2c();
    ~f2c();
};
```
Modeling Complex Systems

- Simulink
- Stateflow
- Embedded MATLAB

- Hybrid Solution

- Hand Code

- Architecture Tool

Algorithmic Complexity

Architectural Complexity

- Simulink + UML (example 1)
- Simulink + AUTOSAR (example 2)

- UML
- AUTOSAR
- AADL
- SCA
- System-C
Projects with Low Algorithm Complexity and High Architectural Complexity

- Example applications:
  - User interfaces
  - Data management

+ means IDE links the code together
Projects with High Algorithm Complexity and Low/Moderate Architectural Complexity

- Example applications:
  - Controllers (avionics, powertrain)
  - Signal processing

+ means IDE links the code together

IDE Tool Chain

Algorithm Code

Non-Algorithmic & Interface Code

Drivers

Legacy Code/S-Function Import

Real-Time Workshop Embedded Coder

Algorithm Modeling

Simulink

Stateflow

Blocksets

Embedded MATLAB

+++
Projects with High Algorithmic and Architectural Complexity

- Example applications:
  - Command, Control, Communications, Computers and Intelligence
  - Tracking systems

+ means IDE links the code together
Example 1: Hybrid Simulink and UML Software Development

**Goals**

- Enable integration of Simulink components into a UML-defined software architecture
- Ensure that code generated from Simulink fits with framework generated from UML

**Role of Simulink**

- Design and test an algorithmic component
- Generate C++ compatible production code

**Role of a UML tool**

- Define the architecture of the software system
- Capture the interface to Simulink generated code
- Generate non-algorithmic code
Mapping a Model to a “Component” Class

1. Build model (or atomic subsystem)
2. Generate code for model/subsystem*
3. Create a class for the subsystem
4. Relate operations to function members in code
5. Relate attributes to data members in code

* Uses automatic C++ encapsulation feature
Automated Code-Level Integration Approach

- **Simulink and Stateflow**
  - Real-Time Workshop Embedded Coder
  - Code from Real-Time Workshop Embedded Coder with C++ option

- **UML Tool**
  - C++ Generator
  - Parse C++ header files into UML classes
  - C++ code from UML tool

- **Parse Code**
  - If Simulink model changes, regenerate the code and re-parse.

- **Code Integration Environment**
  - Compile and link code files from both sources

- **C++**
Example 2: Model-Level Integration with AUTOSAR (Automotive Open System Architecture)

- AUTOSAR Goals
  - Implement and standardize on a single platform as an OEM-wide “Standard Core” solution
  - Enable OEMs/suppliers to focus on added value

- AUTOSAR Key Technologies*
  - Basic Software
    - Software architecture including a complete basic (environmental) software stack for an ECU as an integration platform for hardware-independent software applications
  - Methods of Software Integration
    - Exchange formats (templates) to enable a seamless configuration process of the basic software stack and the integration of application software in ECUs
  - Functional API
    - Specification of functional interfaces as a standard for application software modules

*Source: Helmut Fennel, OOP 2007
AUTOSAR Architecture

- AUTOSAR Software Component
  - Interface
  - ECU Firmware
- Standard Software
- API 2 VFB & RTE relevant
- API 1 RTE relevant
- API 0
- API 3 Private Interfaces inside Basic Software possible

AUTOSAR Runtime Environment (RTE)

- Standardized Interface
  - Operating System
  - Standardized Interface
- Standardized AUTOSAR Interface
  - Services
  - Communication
  - AUTOSAR Interface
- ECU Abstraction
  - Standardized Interface
  - Complex Device Drivers
- Standardized Interface
  - Microcontroller Abstraction

Basic Software

ECU-Hardware
AUTOSAR Target Workflow

Simulink compatible XML subset of AUTOSAR

Put your model here

<xml>
</xml>

<xml>
</xml>

<xml>
</xml>

Export/Code Gen

void runnable(void) {
...
Rte_Read_p_d(&indata);
...
}

Export Specification

Import Specification

Verify

Design

Merge

Optimised code and XML generated from Simulink subsystem with configuration data

Software mapping specified as Simulink configuration data

AUTOSAR System Authoring Tool
Generated Software/Description
Generated Software/Description

Component Description

Internal Behavior Description

Implementation
Generated Software/Description

Component Description

Internal Behavior Description

Implementation

Key
- Provided Sender/Receiver Port
- Required Sender/Receiver Port
- Provided Client/Server Port
- Required Client/Server Port
- Service Port(s)

ASC

.EXE

.c

.h

.oil

.XML

.c

.h
Conclusion

- **Simulink is the established environment for algorithm development**
  - Rich design and verification environment
  - Route to production code

- **Software architectures are becoming more complex**
  - A number of languages are emerging for the description of software architectures
  - Need to present algorithmic models as components for integration
  - Three main activities
    - Characterize an algorithmic model for optimal use in software
    - Adapt an algorithmic model to the demands of the application
    - Publish the adapted model for use in a given platform

- **Challenges in supporting collaboration between software engineer and algorithm designer**
  - Providing a framework within Simulink that allows the software engineer to prepare an algorithm for deployment into a software architecture
  - Maintaining a consistent representation of the algorithm in both environments
  - Generating the required artefacts to support integration with the software engineer’s design flow