Master Class: Target Optimized Code Generation

Shobhit Shanker
Senior Application Engineer-Code Generation & Verification
Today’s Agenda

- Why is Target Optimization Necessary?
- What is Target Optimization?
- Methods to achieve Target Optimization
- Q&A
Why is Target Optimization Necessary?

Will the code generated from the controller be fast enough on the processor?
What does Target Optimization Mean?

- The target specific characteristics are exploited e.g. processor specific libraries
- Target specific library are used of operations
  - For e.g. Texas Instruments provides an optimized run time support library for the C2000 (fastRTS) intended to improve performance of the sin, cos, & division
- Use custom code to replace default Simulink generated code
What happens if I link against an optimized library?

- The controller uses a few “expensive” operations:
  - A \texttt{sin} and \texttt{cos} are used for coordinate transformation
  - A \texttt{division} operation is used for an angle wrap calculation

How much faster would the code execute? 
Would the code outputs be the same?
Can I integrate a processor specific function to replace sin and cos?

- Texas Instruments provides an optimized run time support library for the C2000 (fastRTS) intended to improve performance of the sin, cos, & division

- The optimized library (fastRTS) also provides a function called **sincos** which calculates the sin and cos together to further improve performance

How would I generate a call to **sincos**?
## Integrate processor optimized library

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Controller Output</th>
<th>Max Execution Time</th>
<th>Speed Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSI C</td>
<td>----</td>
<td>7.95 usec</td>
<td>---</td>
</tr>
<tr>
<td><strong>Optimized Library</strong></td>
<td>No Change</td>
<td>6.25 usec</td>
<td><strong>15.7%</strong></td>
</tr>
<tr>
<td>FastRTS: divide, sin(), cos()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Optimized Library + Code Replacement</strong></td>
<td>No Change</td>
<td>6.01 usec</td>
<td><strong>19.8%</strong></td>
</tr>
<tr>
<td>FastRTS: divide, sincos()</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

I can integrate optimizations and assess code performance!
Different Methods of Achieving Target Optimization in Generated Code

- End goal is to generate target specific functions in generated code
- Can be done through
  - Code Replacement Tables
  - Legacy Code Integration
  - Using Embedded Targets (Only for specific microcontrollers/DSP’s)
Hardware Implementation

- Trivially configure the hardware details
Software Environment

- Specify and constrain the software environment
Different Methods of Achieving Target Optimization in Generated Code

- Legacy Code Integration
  - Code Replacement Tables
  - Embedded Targets
Legacy Code Tool Support Summary

- The Legacy Code Tool is an efficient, easy to use, legacy function interfacing tool
- The tool consists of a set of m-functions (APIs) that can be used to generate the required S-function support files

<table>
<thead>
<tr>
<th>Data Type</th>
<th>S-Function I/O</th>
<th>S-Function Parameter</th>
<th>S-Function DWork</th>
</tr>
</thead>
<tbody>
<tr>
<td>Built-in data types</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Enumerated data types</td>
<td>R2010a</td>
<td>R2010a</td>
<td>R2010a</td>
</tr>
<tr>
<td>Simulink.Bus (scalar only)</td>
<td>Y</td>
<td>R2010a</td>
<td>Y</td>
</tr>
<tr>
<td>Simulink.AliasType</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Simulink.NumericType</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Fixed-point data types</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Fi object</td>
<td>N/A</td>
<td>Y</td>
<td>N/A</td>
</tr>
<tr>
<td>Complex number</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Scalar, 1D array and 2D array</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>ND array</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>void* &amp; void**</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

1. Header file that declares the bus/alias must be supplied
2. Header file must be supplied if also an alias
3. Must be an alias or a numeric type
4. Built-in complex data types only
Legacy Code Tool Example

```c
void lct_enum_structparam_step(void)
{
    output1 = foo(&P, &output2);
}
```

- `my_header.h`
- `my_source.c`

> `P.number = MyEnum.TWO`
> `P.value = pi;`

- Structures and Enums supported

```python
def = legacy_code('initialize');
def.SFunctionName = 'sfun_enum_and_structparam';
def.OutputParamSpec = 'MyEnum y1 = foo(MyStruct p1[1], double y2[1]);'
def.HeaderFiles = {'my_header.h'};
def.SourceFiles = {'my_source.c'};
legacy_code('generate_for_sim', def);
```
Different Methods of Achieving Target Optimization in Generated Code

- Legacy Code Integration
- Code Replacement Tables
- Embedded Targets
Code Replacement Tool-CRTOOL

- Replace default arithmetic functions & operations with custom / target specific code
- Code replacement library (CRL) API, which creates code replacement tables
- Optimize target speed and memory and better integrate model code with external and legacy code.
Product demo to create code replacement tables

- Support solution provides an example of \texttt{sincos} code replacement
Different Methods of Achieving Target Optimization in Generated Code

- Legacy Code Integration
- Code Replacement Tables
- Embedded Targets
What can you do with the Target Support Package?

- Target Support Package™ TC2 integrates MATLAB® and Simulink® with TI's eXpressDSP™ tools and C2000™ processors.

- Develop and validate control designs and digital signal processing algorithms from concept through code.

- Automatic processor specific code generation, prototyping, and embedded system deployment on C2000 processors.
Key Product Features

- Automates the execution, testing, verification, and validation of generated code on TI's C2000™ processors

- Enables real-time evaluation of system designs on eZdsp™ F2808, eZdsp™ F2812, and eZdsp™ F28335 boards

- Supports custom boards based on all DSP devices in C280x, C281x, and C2833x families

- Provides block-level access to on-chip peripherals, such as PWM, ADC, CAN, CCP, and target memory

- Provides C280x GPIO Digital Input and Output blocks for input and output capabilities
Verification of Generated Code
Assessing behavior and execution time using Processor-In-the-Loop (PIL)
Verify the behavior and measure resource usage of the generated code

Did the code perform behaviorally correct?

Did the code fit within our resource requirements?

I can reuse my simulation test to assess the code!
Automatic code generation for production ECU

Challenge
Caterpillar wanted to adopt automatic code generation for new algorithms while interfacing to existing development processes, architectures, standards and legacy software

Solution (with the help of MathWorks’ Consulting)
- Customized code generation for interfacing to Caterpillar’s legacy processes and software

Value
- ECU development effort reduced by a factor of 2 to 4, depending on the project
- ECU development calendar time reduced by a factor of greater than 2

“By adopting automatic code generation and with help from MathWorks Consulting, we have been able to reduce ECU software development time and effort by a factor of 2 to 4, depending on the project.” - Larry E. Kendrick, Caterpillar, Inc.

Source: SAE World Congress : 2004-01-0894 – Caterpillar Automatic Code Generation
Thank You