Migrating Motor Controller C++ Software from a Microcontroller to a PolarFire® FPGA with Smart High-Level Synthesis

Introduction

This white paper is aimed at engineers, designing embedded motor controllers for Industrial applications. Specifically, industrial control applications that involve the precise control and coordination of several motors using a software-based motor controller. Embedded motor controller software typically targets an Arm Cortex-M® microcontroller running a real-time operating system. However, certain industrial control applications require higher motor performance and precision than what is possible to achieve using a software-based motor controller.

In closed-loop motor controllers, low latency and low jitter are critical because the motor current must be changed at regular intervals based on real-time sensor feedback from motor encoders. Software running on a microcontroller, even running a real-time operating system, may struggle to achieve low latencies (sub 10 microseconds) due to jitter, interrupt latency, and time taken to compute. Instead, it is proposed to migrate the software-based motor controller to a hardware-based motor controller implemented on a Microchip PolarFire® FPGA. An FPGA implementation offers deterministic latency and jitter, which enables your industrial design to achieve the best possible motor performance and precision.

Designing a new hardware-based motor controller from scratch for an FPGA, using Verilog/VHDL, can be time consuming. Typically, an engineer already has an existing motor controller designed in C/C++ that works on a microcontroller. In this situation, the ideal solution is to automatically convert the existing C++ software code into an equivalent hardware implementation, targeting a Microchip PolarFire FPGA. This is made easy by the Smart High-Level Synthesis (SmartHLS™) tool and integrated development environment, which can compile C++ software into a hardware block targeting a PolarFire FPGA.

By migrating from a microcontroller to an FPGA, the motor controller has a deterministic latency, with a fixed number of clock cycles between receiving motor encoder feedback and driving current to the motors. In this whitepaper, a case-study is given on how an engineer migrates an existing C++ motor control application working on a microcontroller to a Microchip PolarFire FPGA using the SmartHLS tool.

SmartHLS provides an integrated development environment tool that enables engineers to compile C/C++ software into Verilog targeting a Microchip FPGA device, improving productivity and time-to-market. For more information, visit the SmartHLS webpage.

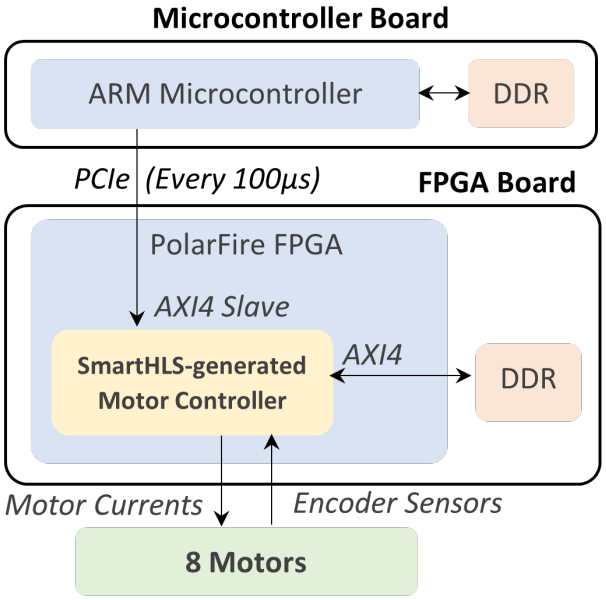
> "SmartHLS allowed us to easily port our existing C++ motor controller code into a Microchip FPGA with minimal modifications. The generated hardware core met our latency, Fmax, and area requirements. The SmartHLS team has also provided us with a great support!" - Koji Yoneda, CEO, Sodick America

Table of Contents

1. Motor Controller System Overview

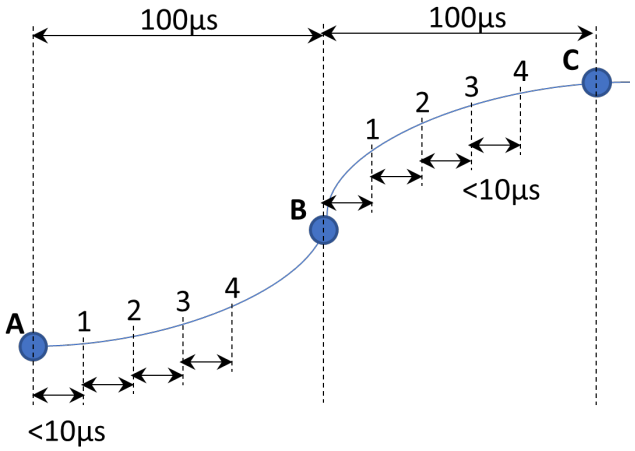
The target system for the motor controller is shown in the following figure consists of two boards. The microcontroller board has an Arm microcontroller with an attached DDR memory, which is connected over PCIe with the FPGA board. The FPGA board has a PolarFire FPGA (MPF300TS_ES-1FCG1152E) connected to 8 motors and has a DDR memory holding data needed by the motor control algorithm.

Figure 1-1. Motor Controller System Diagram



The following figure, shows how the coarse-grained motor positions are calculated by the microcontroller every 100 microseconds (labeled A, B and C). Every 100 microseconds, the Arm microcontroller sends the updated position of the motor over the PCIe interface, updates the FPGA motor controller control status registers, and starts the fine-grained motor controller running on the FPGA.

Figure 1-2. Coarse and Fine-Grained Motor Path Calculation



On the FPGA, the SmartHLS-generated motor controller hardware core is controlled via an AXI slave interface. The fine-grained motor positions are calculated by the FPGA every 10 microseconds or less (labeled as 1, 2, 3, 4 in the preceding figure) using input feedback from the motor encoder sensors. The FPGA outputs the motor currents every 10 microseconds to adjust the motor positions.

The FPGA motor algorithm in C++ consists of two loops iterating over 8 motors: the path generator loop and the motor output current loop. There is also an initial stage to perform a DMA request for the compensation data from DDR memory using an AXI master interface. SmartHLS synthesizes the C++ code into a hardware core, including the AXI slave and master interfaces.

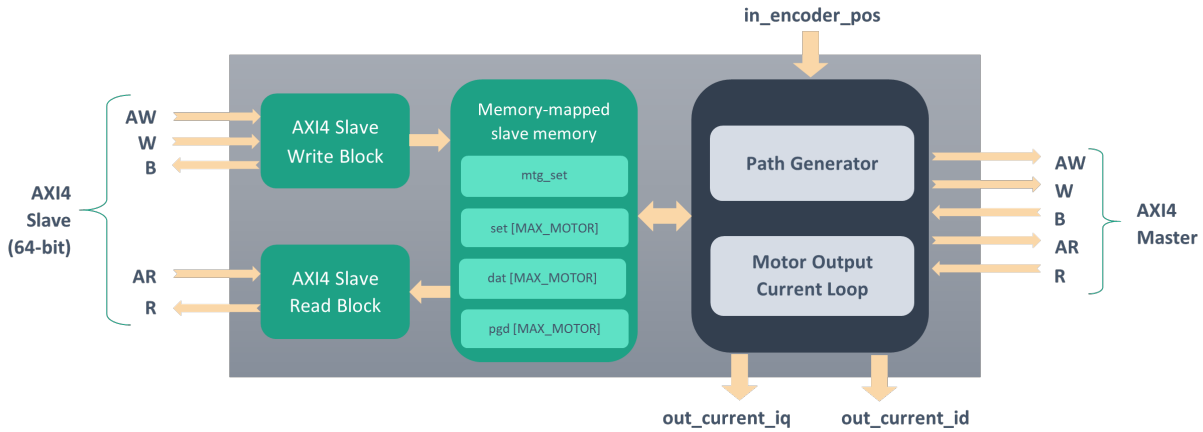
1.1 SmartHLS Design Methodology

You can use the SmartHLS eclipse-based IDE to implement their motor controller software in C++. First, the engineer verifies their application's functionality by compiling and running the software on their local computer with a C++ software test bench. Next, the engineer specifies the circuit F_{max} constraints, and uses SmartHLS to automatically convert the C++ software into an equivalent Verilog module in a few minutes.

For hardware verification, SmartHLS supports an automated cycle-accurate co-simulation flow using Modelsim. The co-simulation flow runs the C++ software test bench to gather input test vectors and expected outputs, then simulates the generated Verilog module with the input test vectors, while verifying the outputs. The co-simulation flow also allows you to accurately measure the cycle latency of the motor controller and includes support for AXI interfaces. Finally, you can click a button in the SmartHLS IDE to synthesize the design using Microchip Libero®, generate an FPGA bit stream and report the FMax and area.

The following figure shows a block diagram of the SmartHLS-generated motor controller hardware block. The input to the controller is the motor position from the encoder sensor (encoder_pos). For each of the 8 motors, there are two current outputs (current_iq, and current_id), which are represented in the C++ as arrays. The motor controller includes an AXI4 slave interface, to allow the Arm microcontroller to burst write data to the control status registers in memory-mapped slave memory. The memory-mapped slave memory is represented in C++ as a global struct. The motor controller also contains an AXI4 master interface to burst read from DDR off-chip memory. The core motor controller algorithm consists of two loops that iterate over the 8 motors: the path generator loop and the motor output current loop. The SmartHLS loop pipelining user-constraint is applied so that the loops can be pipelined with overlapping execution of iterations for better latency.

Figure 1-3. SmartHLS Generated Motor Controller FPGA Hardware Block



1.2 SmartHLS Hardware Interfaces

In SmartHLS, you can specify the top-level hardware interfaces using function arguments with the standard C++ types and SmartHLS-specific C++ types for AXI interfaces. These types are compiled into the corresponding hardware interface in Verilog.

The following example shows the motor controller C++ top-level interfaces for the motor controller hardware block in C++. For each motor, the block has two 16-bit outputs (out_current_iq and out_current_id) and a 32-bit input (in_encoder_pos). The inputs and outputs are represented as an array in C++, which is split into individual ports when generating the Verilog module. The AXI master interface is specified as a reference to the SmartHLS-provided "AxiInterface" C++ class, with template arguments specifying an address width of 32 bits, data width of 64 bits, and byte enable of 8 bits. This can be used to DMA data from DDR memory using burst requests.

Example 1-1. C++ Top-Level for Motor Controller Block

```
void __attribute__((noinline)) MotorControl
(
    uint32 in_encoder_pos[MOTORS],
    int16 out_current_iq[MOTORS],
    int16 out_current_id[MOTORS],
    AxiInterface<uint32, uint64, uint8> &master
) {
```

The AXI slave memory-mapped memory is represented in C++ as a global struct as shown in the following example. SmartHLS has a user-constraint to specify that a struct must be accessible from an AXI slave interface. SmartHLS then automatically generates the 64-bit AXI slave interface ports and assign addresses to the struct elements. These memory-mapped addresses can be found in a generated header file.

Example 1-2. AXI4 Slave Memory-Mapped C++ Struct

```
struct SlaveMemoryT {
 HardwareMtmGlobalSettings mtg_set;
 HardwareMtmGlobalData mtg_dat;
 HardwareMotorSettings set[MOTORS];
 HardwareMtmCommonData dat[MOTORS];
 PathGenData pgd[MOTORS];
 ...
}
```

The following example lists the Verilog module interface generated by SmartHLS for the 8 motor inputs or outputs. Input ports are labeled with "_readdata" and output ports are labeled with "_writedata". Whether a port is an input or output is inferred automatically by SmartHLS based on how the C++ array function arguments are accessed in the top-level function. Arrays are only read from becoming the input ports.

Example 1-3. SmartHLS-Generated Verilog for Motor Inputs or Outputs

```
output reg [31:0] master_AW_AWADDR_data_to_sink;
input master_AW_ready_from_sink;
output reg master_AW_valid_to_sink;
output reg [7:0] master_AW_AWLEN_data_to_sink;
output reg [63:0] master_W_WDATA_data_to_sink;
input master_W_ready_from_sink;
...
input [31:0] axi_s_AW_AWADDR_data_from_source;
output reg axi_s_AW_ready_to_source;
input axi_s_AW_valid_from_source;
input [7:0] axi_s_AW_AWLEN_data_from_source;
input [63:0] axi_s_W_WDATA_data_from_source;
output reg axi_s_W_ready_to_source;
```

The following example lists the generated Verilog for the AXI master and slave interfaces. These are generated automatically from the C++ software and the user-specified constraints.

Example 1-4. SmartHLS-Generated Verilog for AXI4 Master or Slave Interface

```
input [31:0] in_enc_pos_a0_readdata;
input [31:0] in_enc_pos_a1_readdata;
...
input [31:0] in_enc_pos_a7_readdata;
output reg [15:0] out_current_iq_a0_writedata;
output reg [15:0] out_current_id_a0_writedata;
output reg [15:0] out_current_iq_a1_writedata;
output reg [15:0] out_current_id_a1_writedata;
...
output reg [15:0] out_current_iq_a7_writedata;
output reg [15:0] out_current_id_a7_writedata;
SmartHLS-generated Verilog for AXI4 Master/Slave Interface
```

1.3 Floating-Point and Fixed-Point Support

The motor controller C++ software algorithm uses the standard floating-point operations. SmartHLS includes support for floating-point IP cores that are optimized for the PolarFire FPGA DSP architecture both in terms of latency and FMax. SmartHLS also supports constraining the number of floating-point hardware units used for each operation type. These floating-point units are shared among the required floating-point operations, which saves FPGA area but can reduce performance.

The following example shows an example of the C++ template arguments for the SmartHLS floating-point multiplier core.

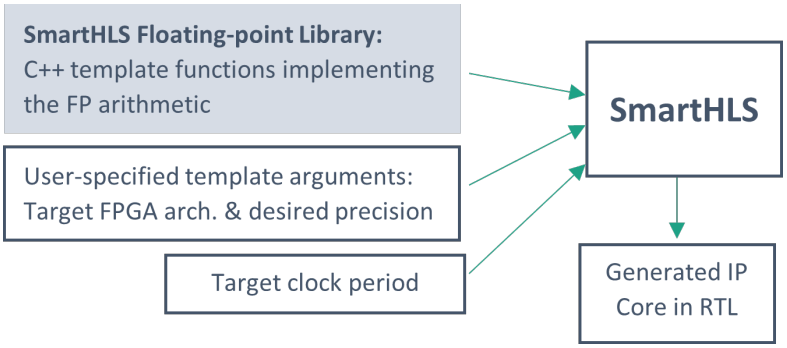
Example 1-5. Floating-Point Multiplier Template Arguments

```
template <
    // Configure floating-point format:
    unsigned M_W,     // Mantissa width
    unsigned E_W,     // Exponent width
    class FPType = ap_fp<M_W, E_W>,

    // Configure DSP multiplier width: A x B
    unsigned MULT_W_A,     // Input A width
    unsigned MULT_W_B,     // Input B width
    >
FPType fmult(FPType A, FPType B) { ... }
```

By default, the floating-point cores are configured automatically by SmartHLS based on the C++ floating point types. But, if you wish to specify custom mantissa and exponent widths, for example in an AI application, you can directly instantiate these floating-point classes in your C++ code.

Figure 1-4. SmartHLS Floating-point Library



SmartHLS also provides an easy way for you to specify fixed-point operations using the C++ fixed-point library (ap_fixed). SmartHLS can support all major C++ operations in fixed-point. The fixed-point library allows you to start with a floating-point implementation of the motor controller algorithm to match the microcontroller implementation, then migrate to fixed-point math if required to reduce the area on the FPGA. Fixed-point hardware units consume significantly less area than floating point hardware units, while offering higher performance but more upfront design effort.

The following table lists few examples of the SmartHLS ap_fixed types with their corresponding ranges.

Table 1-1. Example SmartHLS Fixed-Point Types and Ranges

Type	Quantum	Range
ap_fixpt<8, 4>	0.0625	-8 to 7.9375
ap_ufixpt<4, 12>	256	0 to 3840
ap_ufixpt<4, -2>	0.015625	0 to 0.234375

1.4 FPGA Performance

The initial implementation of the motor controller C++ used a push button approach to SmartHLS. Next, weeks were spent performing design space exploration using SmartHLS user-constraints and tuning the Microchip Libero synthesis options.

The following table lists the hardware quality of results for the motor controller hardware targeting a PolarFire FPGA (MPF300TS_ES-1FCG1152E). The cycle latency is almost halved and the F_{max} is improved to meet the 200 MHz target clock frequency. The final deterministic latency for the motor controller is approximately 2 µs. The area of the motor controller is also reduced to fit inside the 300 K LUT PolarFire FPGA.

Table 1-2. Motor Controller FPGA Hardware Quality of Results

	Cycle Latency	Clock Frequency	Deterministic Latency	LUT Count
Initial	800	150 MHz	5.33 µs	320K
Optimized	**450**	**210 MHz**	**2.14 µs**	**200K**

Experiments were performed comparing the original C++ motor controller to the improved FPGA version. It was found that the FPGA-based motor controller had a 2.5-6x speedup in terms of latency compared to the Arm microcontroller, depending on the jitter of the microcontroller and real-time operating system.

2. Conclusion

In conclusion, SmartHLS can offer an easy path to migrate existing C/C++ code targeting a micro-controller to a Microchip PolarFire FPGA. For industrial control applications, an FPGA can offer motor controllers with significantly better deterministic latency and jitter compared to a microcontroller.

2.1 Key Takeaways

- SmartHLS simplifies FPGA design by allowing you to program the FPGA using C/C++ software.
- C/C++ code for microcontrollers can be migrated to a Microchip FPGA using SmartHLS.
- Microchip FPGAs offer 2.5-6X better latency/jitter for motor control vs. a microcontroller.

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods being used in attempts to breach the code protection features of the Microchip devices. We believe that these methods require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Attempts to breach these code protection features, most likely, cannot be accomplished without violating Microchip's intellectual property rights.
- Microchip is willing to work with any customer who is concerned about the integrity of its code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is "unbreakable." Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Trademarks

ISBN: 978-1-5224-8704-3

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: www.microchip.com/support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4485-5910 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-72400 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-72884388 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820