ALF in C++

A guide for porting a Eclipse alf C project to a C++ Project.

Frank Blaauw & Spencer Shaw
10-6-2009

Alf in C++

By Spencer Shaw and Frank Blaauw.

This guide is written for development on the Playstation 3 and describes how to use ALF in a C++ project. For this document the IBM developerWorks forums were used.

In no event will the UMCG nor the writers of this document be liable for any damage arising directly or indirectly from any use of the information obtained from this document.

This guide and many other guides can be downloaded from:
http://code.google.com/p/fedora-cell-project/

If you encounter any errors on using this document, please read the inform us via the google-code page or google-group.

Table of Contents

1. Requirements for installation

Everything you might need for this installation is described below.

1.1 Minimal Requirements.

- ✓ A computer
- ✓ Distribution of Fedora Core 9 (LiveCD)
- ✓ IBM SDK for Multicore acceleration installed (With ALF)
- ✓ Eclipse

For this guide we used the LiveCD because the DVD installer would not boot, because of a strange slash in the name of our hard-disk which messed everything up, using this live CD did not gave us this problem.

2. Creating the projects

This chapter will provide you with the information you need use your Accelerated Library Framework in a C++ project.

For this example we have to create 3 projects:

Cell PPU Executable (we call it ppu_Cpptest) (PPU GNU 32 Bit Debug Tool Chain)
Cell SPU Executable (we call it spu_Cpptest) (SPU GNU Debug Tool Chain)
Cell PPU Shared Library (we call it libCpptest) (PPU GNU 32 Bit Debug Tool Chain)

3. Compiler settings

3.1 The Shared Library

1. Add reference from libCpptest to spu_Cpptest.
2. Add inputs -> embedded SPU inputs:

   "${workspace_loc:/spu_Cpptest/spu-gnu-debug/spu_Cpptest}"

3.2 The PPU Project

1. Create a new Cell PPU Executable project (PPU GNU 32 Bit Debug Tool Chain). In this example we call it ppu_Cpptest. Installing Fedora

2. Add the following libraries to the C++ linker:

   /opt/cell/sysroot/usr/lib (-L/opt/cell/sysroot/usr/lib)
   dl (-ldl)
   pthread (-lpthread)
   spe2 (-lspe2)
   alf (-lalf)

3. Add the following libraries to the C and C++ compiler with debug options:

   /opt/cell/sysroot/usr/include (-I/opt/cell/sysroot/usr/include)
   /opt/cell/sysroot/opt/cell/sdk/usr/include (-I/opt/cell/sysroot/opt/cell/sdk/usr/include)

Add a reference from ppu_Cpptest to libCpptest.

3.3 The SPU Project

1. Create a new Cell SPU Executable project (SPU GNU Debug Tool Chain). In this example we call it spu_Cpptest. Installing Fedora

2. Add the following libraries to the C++ linker:

/opt/cell/sysroot/usr/spu/lib (-L/opt/cell/sysroot/usr/spu/lib)
alf (-lalf)

3. Add the following libraries to the C and C++ compiler with debug options:

   /opt/cell/sysroot/usr/spu/include (-I/opt/cell/sysroot/usr/spu/include)
   "/root/workspace/ppu_Cpptest" (-I"/root/workspace/ppu_Cpptest")

4. Add to Paths and Symbols -> Includes -> GNU C and C++: /ppu_Cpptest

4. The Example code:

These code examples are all (slightly edited) versions of the code the ALF Template wizard generates.

4.1 The SPU Code

```
//spu_Cpptest.c: source file for accelerator node program
#include <alf_accel.h>
#include <stdio.h>
#include "common.h"

extern "C" int comp_kernel(void *p_task_context,
            void *p_parm_ctx_buffer,
            void *p_input_buffer,
            void *p_output_buffer,
            void *p_inout_buffer,
            unsigned int current_count,
            unsigned int total_count)
{

            my_param_t *p_param = (my_param_t *) p_parm_ctx_buffer;

            //TODO: Code your computing kernel here
            printf("%s\n", p_param->c.msg);

            return 0;
}


extern "C" int input_prepare(void* p_task_context __attribute__ ((unused)),
               void *p_parm_ctx_buffer,
```

```
            void *p_dtl,
            unsigned int current_count __attribute__ ((unused)),
            unsigned int total_count __attribute__ ((unused)))
{
        //TODO: Code your prepare_input_list here

        return 0;
}

extern "C" int output_prepare(void *p_task_context,
            void *p_parm_ctx_buffer,
            void *p_dtl,
            unsigned int current_count,
            unsigned int total_count)
{
        //TODO: Code your prepare_input_list here

        return 0;
}

extern "C" int task_context_setup(void *p_task_context)
{
        //TODO: Code your task_context_setup here

        return 0;
}

extern "C" int task_context_merge(void* p_dst_task_context, void* p_task_context)
{
        //TODO: Code your task_context_merge here

        return 0;
}

ALF_ACCEL_EXPORT_API_LIST_BEGIN
        ALF_ACCEL_EXPORT_API("kernel", comp_kernel)
        ALF_ACCEL_EXPORT_API("input", input_prepare)
        ALF_ACCEL_EXPORT_API("output", output_prepare)
        ALF_ACCEL_EXPORT_API("ctx setup", task_context_setup)
        ALF_ACCEL_EXPORT_API("ctx merge", task_context_merge)
ALF_ACCEL_EXPORT_API_LIST_END
```

4.2 The PPU Code

```
//ppu_Ctest.c: source file for host node program.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <libgen.h>
#include <unistd.h>
#include <alf.h>
#include "common.h"
#include "ppu_Cpptest.h"

#define MY_ALIGN(_my_var_type, _my_var_def_, _my_al_) _my_var_type _my_var_def_ \
__attribute__((__aligned__(_my_al_)))

#define PATH_BUF_SIZE 1024              // Max length of path to PPU image

char spu_image_path[] = ".";            // Used to hold the complete path to SPU image
char library_name[PATH_BUF_SIZE + 1];   // Used to hold the name of spu library
char spu_image_name[] = "spu_Cpptest";
char kernel_name[]    = "comp_kernel";
char input_dtl_name[] = "input_prepare";
char output_dtl_name[] = "output_prepare";
char ctx_setup_name[] = "task_context_setup";
char ctx_merge_name[] = "task_context_merge";


int main(int argc, char * argv[])
{
        alf_sys_config_t_CBEA cell_config_parms;
        alf_handle_t alf_handle;
        alf_task_desc_handle_t desc_info_handle;
        alf_task_handle_t task_handle;
        alf_wb_handle_t wb_handle;
        unsigned int nodes;
        int err;
        char *dir = NULL;
        MY_ALIGN(my_param_t, param, 128);


        // Absolute path for spu .so file
        dir = dirname(argv[0]);

        if (dir == NULL)
        {
                fprintf(stderr, "Error in query working directory.\n");
                cell_config_parms.library_path = spu_image_path;
        }
        else
        {
```

```
            cell_config_parms.library_path = dir;
        }

        sprintf(library_name, "liblibCpptest.so");

        alf_init(&cell_config_parms, &alf_handle);
        if ((err = alf_query_system_info(alf_handle, ALF_QUERY_NUM_ACCEL, ALF_ACCEL_TYPE_SPE,
&nodes)) < 0)
        {
            fprintf(stderr, "Error in query system information.\n");
            return (-1);
        }

        alf_num_instances_set (alf_handle, nodes);
        alf_task_desc_create (alf_handle, ALF_ACCEL_TYPE_SPE, &desc_info_handle);
        alf_task_desc_set_int32 (desc_info_handle, ALF_TASK_DESC_WB_PARM_CTX_BUF_SIZE,
sizeof(my_param_t));
        alf_task_desc_set_int32 (desc_info_handle, ALF_TASK_DESC_WB_IN_BUF_SIZE, 0);
        alf_task_desc_set_int32 (desc_info_handle, ALF_TASK_DESC_WB_OUT_BUF_SIZE, 0);
        alf_task_desc_set_int32 (desc_info_handle, ALF_TASK_DESC_WB_INOUT_BUF_SIZE, 0);
        alf_task_desc_set_int32 (desc_info_handle, ALF_TASK_DESC_NUM_DTL_ENTRIES, 1);
        alf_task_desc_set_int32 (desc_info_handle, ALF_TASK_DESC_TSK_CTX_SIZE, 0);
        alf_task_desc_set_int32 (desc_info_handle, ALF_TASK_DESC_MAX_STACK_SIZE, 10);
#ifdef ACCELERATOR_PARTITION
        alf_task_desc_set_int32 (desc_info_handle, ALF_TASK_DESC_PARTITION_ON_ACCEL, 1);
#else
        alf_task_desc_set_int32 (desc_info_handle, ALF_TASK_DESC_PARTITION_ON_ACCEL, 0);
#endif
        alf_task_desc_set_int64 (desc_info_handle, ALF_TASK_DESC_ACCEL_IMAGE_REF_L, (unsigned
long long)spu_image_name);
        alf_task_desc_set_int64 (desc_info_handle, ALF_TASK_DESC_ACCEL_LIBRARY_REF_L, (unsigned
long long)library_name);
        alf_task_desc_set_int64 (desc_info_handle, ALF_TASK_DESC_ACCEL_KERNEL_REF_L, (unsigned
long long)kernel_name);
        alf_task_desc_set_int64 (desc_info_handle, ALF_TASK_DESC_ACCEL_INPUT_DTL_REF_L,
(unsigned long long)input_dtl_name);
        alf_task_desc_set_int64 (desc_info_handle, ALF_TASK_DESC_ACCEL_OUTPUT_DTL_REF_L,
(unsigned long long)output_dtl_name);
        alf_task_desc_set_int64 (desc_info_handle, ALF_TASK_DESC_ACCEL_CTX_SETUP_REF_L,
(unsigned long long)ctx_setup_name);
        alf_task_desc_set_int64 (desc_info_handle, ALF_TASK_DESC_ACCEL_CTX_MERGE_REF_L,
(unsigned long long)ctx_merge_name);

        alf_task_create(desc_info_handle, NULL, 0, 0, 1, &task_handle);

        strcpy(param.c.msg, "Hello World!");
        alf_wb_create(task_handle, ALF_WB_SINGLE, 0, &wb_handle);
        alf_wb_parm_add(wb_handle, (void *)(&param), sizeof(my_param_t), ALF_DATA_BYTE, 0);
        alf_wb_enqueue(wb_handle);
```

```
        alf_task_finalize (task_handle);

        //TODO: Code your application here


        alf_task_wait(task_handle, -1);
        alf_task_destroy(task_handle);
        alf_exit(alf_handle, ALF_EXIT_POLICY_WAIT, -1);

        return 0;
}
```

4.3 The Shared Library.

The shared object project does not contain any code. the files in this folder will be created if the other projects are built correctly.