// ==============================================================

// File generated by Vivado(TM) HLS - High-Level Synthesis from C, C++ and SystemC

// Version: 2017.2

// Copyright (C) 1986-2017 Xilinx, Inc. All Rights Reserved.

// ==============================================================

#include <systemc>

#include <iostream>

#include <cstdlib>

#include <cstddef>

#include <stdint.h>

#include "SysCFileHandler.h"

#include "ap\_int.h"

#include "ap\_fixed.h"

#include <complex>

#include <stdbool.h>

#include "autopilot\_cbe.h"

#include "ap\_stream.h"

#include "hls\_stream.h"

#include "hls\_half.h"

#include "hls\_signal\_handler.h"

using namespace std;

using namespace sc\_core;

using namespace sc\_dt;

// [dump\_struct\_tree [build\_nameSpaceTree] dumpedStructList] ---------->

// [dump\_enumeration [get\_enumeration\_list]] ---------->

// wrapc file define: "a"

#define AUTOTB\_TVIN\_a "../tv/cdatafile/c.simple.autotvin\_a.dat"

// wrapc file define: "b"

#define AUTOTB\_TVIN\_b "../tv/cdatafile/c.simple.autotvin\_b.dat"

// wrapc file define: "c"

#define AUTOTB\_TVOUT\_c "../tv/cdatafile/c.simple.autotvout\_c.dat"

#define AUTOTB\_TVIN\_c "../tv/cdatafile/c.simple.autotvin\_c.dat"

// wrapc file define: "ap\_return"

#define AUTOTB\_TVOUT\_ap\_return "../tv/cdatafile/c.simple.autotvout\_ap\_return.dat"

#define INTER\_TCL "../tv/cdatafile/ref.tcl"

// tvout file define: "c"

#define AUTOTB\_TVOUT\_PC\_c "../tv/rtldatafile/rtl.simple.autotvout\_c.dat"

// tvout file define: "ap\_return"

#define AUTOTB\_TVOUT\_PC\_ap\_return "../tv/rtldatafile/rtl.simple.autotvout\_ap\_return.dat"

class INTER\_TCL\_FILE {

public:

INTER\_TCL\_FILE(const char\* name) {

mName = name;

a\_depth = 0;

b\_depth = 0;

c\_depth = 0;

ap\_return\_depth = 0;

trans\_num =0;

}

~INTER\_TCL\_FILE() {

mFile.open(mName);

if (!mFile.good()) {

cout << "Failed to open file ref.tcl" << endl;

exit (1);

}

string total\_list = get\_depth\_list();

mFile << "set depth\_list {\n";

mFile << total\_list;

mFile << "}\n";

mFile << "set trans\_num "<<trans\_num<<endl;

mFile.close();

}

string get\_depth\_list () {

stringstream total\_list;

total\_list << "{a " << a\_depth << "}\n";

total\_list << "{b " << b\_depth << "}\n";

total\_list << "{c " << c\_depth << "}\n";

total\_list << "{ap\_return " << ap\_return\_depth << "}\n";

return total\_list.str();

}

void set\_num (int num , int\* class\_num) {

(\*class\_num) = (\*class\_num) > num ? (\*class\_num) : num;

}

public:

int a\_depth;

int b\_depth;

int c\_depth;

int ap\_return\_depth;

int trans\_num;

private:

ofstream mFile;

const char\* mName;

};

extern "C" float simple (

float a[8],

float b[8],

float c[8]);

extern "C" float AESL\_WRAP\_simple (

float a[8],

float b[8],

float c[8])

{

refine\_signal\_handler();

fstream wrapc\_switch\_file\_token;

wrapc\_switch\_file\_token.open(".hls\_cosim\_wrapc\_switch.log");

int AESL\_i;

if (wrapc\_switch\_file\_token.good())

{

CodeState = ENTER\_WRAPC\_PC;

static unsigned AESL\_transaction\_pc = 0;

string AESL\_token;

string AESL\_num;

static AESL\_FILE\_HANDLER aesl\_fh;

float AESL\_return;

// output port post check: "c"

aesl\_fh.read(AUTOTB\_TVOUT\_PC\_c, AESL\_token); // [[transaction]]

if (AESL\_token != "[[transaction]]")

{

exit(1);

}

aesl\_fh.read(AUTOTB\_TVOUT\_PC\_c, AESL\_num); // transaction number

if (atoi(AESL\_num.c\_str()) == AESL\_transaction\_pc)

{

aesl\_fh.read(AUTOTB\_TVOUT\_PC\_c, AESL\_token); // data

sc\_bv<32> \*c\_pc\_buffer = new sc\_bv<32>[8];

int i = 0;

while (AESL\_token != "[[/transaction]]")

{

bool no\_x = false;

bool err = false;

// search and replace 'X' with "0" from the 1st char of token

while (!no\_x)

{

size\_t x\_found = AESL\_token.find('X');

if (x\_found != string::npos)

{

if (!err)

{

cerr << "WARNING: [SIM 212-201] RTL produces unknown value 'X' on port 'c', possible cause: There are uninitialized variables in the C design." << endl;

err = true;

}

AESL\_token.replace(x\_found, 1, "0");

}

else

{

no\_x = true;

}

}

no\_x = false;

// search and replace 'x' with "0" from the 3rd char of token

while (!no\_x)

{

size\_t x\_found = AESL\_token.find('x', 2);

if (x\_found != string::npos)

{

if (!err)

{

cerr << "WARNING: [SIM 212-201] RTL produces unknown value 'X' on port 'c', possible cause: There are uninitialized variables in the C design." << endl;

err = true;

}

AESL\_token.replace(x\_found, 1, "0");

}

else

{

no\_x = true;

}

}

// push token into output port buffer

if (AESL\_token != "")

{

c\_pc\_buffer[i] = AESL\_token.c\_str();

i++;

}

aesl\_fh.read(AUTOTB\_TVOUT\_PC\_c, AESL\_token); // data or [[/transaction]]

if (AESL\_token == "[[[/runtime]]]" || aesl\_fh.eof(AUTOTB\_TVOUT\_PC\_c))

{

exit(1);

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if (i > 0)

{

// RTL Name: c

{

// bitslice(31, 0)

// {

// celement: c(31, 0)

// {

sc\_lv<32>\* c\_lv0\_0\_7\_1 = new sc\_lv<32>[8];

// }

// }

// bitslice(31, 0)

{

int hls\_map\_index = 0;

// celement: c(31, 0)

{

// carray: (0) => (7) @ (1)

for (int i\_0 = 0; i\_0 <= 7; i\_0 += 1)

{

if (&(c[0]) != NULL) // check the null address if the c port is array or others

{

c\_lv0\_0\_7\_1[hls\_map\_index].range(31, 0) = sc\_bv<32>(c\_pc\_buffer[hls\_map\_index].range(31, 0));

hls\_map\_index++;

}

}

}

}

// bitslice(31, 0)

{

int hls\_map\_index = 0;

// celement: c(31, 0)

{

// carray: (0) => (7) @ (1)

for (int i\_0 = 0; i\_0 <= 7; i\_0 += 1)

{

// sub : i\_0

// ori\_name : c[i\_0]

// sub\_1st\_elem : 0

// ori\_name\_1st\_elem : c[0]

// output\_left\_conversion : \*(int\*)&c[i\_0]

// output\_type\_conversion : (c\_lv0\_0\_7\_1[hls\_map\_index]).to\_uint64()

if (&(c[0]) != NULL) // check the null address if the c port is array or others

{

\*(int\*)&c[i\_0] = (c\_lv0\_0\_7\_1[hls\_map\_index]).to\_uint64();

hls\_map\_index++;

}

}

}

}

}

}

// release memory allocation

delete [] c\_pc\_buffer;

}

// output port post check: "ap\_return"

aesl\_fh.read(AUTOTB\_TVOUT\_PC\_ap\_return, AESL\_token); // [[transaction]]

if (AESL\_token != "[[transaction]]")

{

exit(1);

}

aesl\_fh.read(AUTOTB\_TVOUT\_PC\_ap\_return, AESL\_num); // transaction number

if (atoi(AESL\_num.c\_str()) == AESL\_transaction\_pc)

{

aesl\_fh.read(AUTOTB\_TVOUT\_PC\_ap\_return, AESL\_token); // data

sc\_bv<32> ap\_return\_pc\_buffer;

int i = 0;

while (AESL\_token != "[[/transaction]]")

{

bool no\_x = false;

bool err = false;

// search and replace 'X' with "0" from the 1st char of token

while (!no\_x)

{

size\_t x\_found = AESL\_token.find('X');

if (x\_found != string::npos)

{

if (!err)

{

cerr << "WARNING: [SIM 212-201] RTL produces unknown value 'X' on port 'ap\_return', possible cause: There are uninitialized variables in the C design." << endl;

err = true;

}

AESL\_token.replace(x\_found, 1, "0");

}

else

{

no\_x = true;

}

}

no\_x = false;

// search and replace 'x' with "0" from the 3rd char of token

while (!no\_x)

{

size\_t x\_found = AESL\_token.find('x', 2);

if (x\_found != string::npos)

{

if (!err)

{

cerr << "WARNING: [SIM 212-201] RTL produces unknown value 'X' on port 'ap\_return', possible cause: There are uninitialized variables in the C design." << endl;

err = true;

}

AESL\_token.replace(x\_found, 1, "0");

}

else

{

no\_x = true;

}

}

// push token into output port buffer

if (AESL\_token != "")

{

ap\_return\_pc\_buffer = AESL\_token.c\_str();

i++;

}

aesl\_fh.read(AUTOTB\_TVOUT\_PC\_ap\_return, AESL\_token); // data or [[/transaction]]

if (AESL\_token == "[[[/runtime]]]" || aesl\_fh.eof(AUTOTB\_TVOUT\_PC\_ap\_return))

{

exit(1);

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if (i > 0)

{

// RTL Name: ap\_return

{

// bitslice(31, 0)

// {

// celement: return(31, 0)

// {

sc\_lv<32> return\_lv0\_0\_1\_0;

// }

// }

// bitslice(31, 0)

{

// celement: return(31, 0)

{

// carray: (0) => (1) @ (0)

{

if (&(AESL\_return) != NULL) // check the null address if the c port is array or others

{

return\_lv0\_0\_1\_0.range(31, 0) = sc\_bv<32>(ap\_return\_pc\_buffer.range(31, 0));

}

}

}

}

// bitslice(31, 0)

{

// celement: return(31, 0)

{

// carray: (0) => (1) @ (0)

{

// sub :

// ori\_name : AESL\_return

// sub\_1st\_elem :

// ori\_name\_1st\_elem : AESL\_return

// output\_left\_conversion : \*(int\*)&AESL\_return

// output\_type\_conversion : (return\_lv0\_0\_1\_0).to\_uint64()

if (&(AESL\_return) != NULL) // check the null address if the c port is array or others

{

\*(int\*)&AESL\_return = (return\_lv0\_0\_1\_0).to\_uint64();

}

}

}

}

}

}

}

AESL\_transaction\_pc++;

return AESL\_return;

}

else

{

CodeState = ENTER\_WRAPC;

static unsigned AESL\_transaction;

static AESL\_FILE\_HANDLER aesl\_fh;

// "a"

char\* tvin\_a = new char[50];

aesl\_fh.touch(AUTOTB\_TVIN\_a);

// "b"

char\* tvin\_b = new char[50];

aesl\_fh.touch(AUTOTB\_TVIN\_b);

// "c"

char\* tvin\_c = new char[50];

aesl\_fh.touch(AUTOTB\_TVIN\_c);

char\* tvout\_c = new char[50];

aesl\_fh.touch(AUTOTB\_TVOUT\_c);

// "ap\_return"

char\* tvout\_ap\_return = new char[50];

aesl\_fh.touch(AUTOTB\_TVOUT\_ap\_return);

CodeState = DUMP\_INPUTS;

static INTER\_TCL\_FILE tcl\_file(INTER\_TCL);

int leading\_zero;

// [[transaction]]

sprintf(tvin\_a, "[[transaction]] %d\n", AESL\_transaction);

aesl\_fh.write(AUTOTB\_TVIN\_a, tvin\_a);

sc\_bv<32>\* a\_tvin\_wrapc\_buffer = new sc\_bv<32>[8];

// RTL Name: a

{

// bitslice(31, 0)

{

int hls\_map\_index = 0;

// celement: a(31, 0)

{

// carray: (0) => (7) @ (1)

for (int i\_0 = 0; i\_0 <= 7; i\_0 += 1)

{

// sub : i\_0

// ori\_name : a[i\_0]

// sub\_1st\_elem : 0

// ori\_name\_1st\_elem : a[0]

// regulate\_c\_name : a

// input\_type\_conversion : \*(int\*)&a[i\_0]

if (&(a[0]) != NULL) // check the null address if the c port is array or others

{

sc\_lv<32> a\_tmp\_mem;

a\_tmp\_mem = \*(int\*)&a[i\_0];

a\_tvin\_wrapc\_buffer[hls\_map\_index].range(31, 0) = a\_tmp\_mem.range(31, 0);

hls\_map\_index++;

}

}

}

}

}

// dump tv to file

for (int i = 0; i < 8; i++)

{

sprintf(tvin\_a, "%s\n", (a\_tvin\_wrapc\_buffer[i]).to\_string(SC\_HEX).c\_str());

aesl\_fh.write(AUTOTB\_TVIN\_a, tvin\_a);

}

tcl\_file.set\_num(8, &tcl\_file.a\_depth);

sprintf(tvin\_a, "[[/transaction]] \n");

aesl\_fh.write(AUTOTB\_TVIN\_a, tvin\_a);

// release memory allocation

delete [] a\_tvin\_wrapc\_buffer;

// [[transaction]]

sprintf(tvin\_b, "[[transaction]] %d\n", AESL\_transaction);

aesl\_fh.write(AUTOTB\_TVIN\_b, tvin\_b);

sc\_bv<32>\* b\_tvin\_wrapc\_buffer = new sc\_bv<32>[8];

// RTL Name: b

{

// bitslice(31, 0)

{

int hls\_map\_index = 0;

// celement: b(31, 0)

{

// carray: (0) => (7) @ (1)

for (int i\_0 = 0; i\_0 <= 7; i\_0 += 1)

{

// sub : i\_0

// ori\_name : b[i\_0]

// sub\_1st\_elem : 0

// ori\_name\_1st\_elem : b[0]

// regulate\_c\_name : b

// input\_type\_conversion : \*(int\*)&b[i\_0]

if (&(b[0]) != NULL) // check the null address if the c port is array or others

{

sc\_lv<32> b\_tmp\_mem;

b\_tmp\_mem = \*(int\*)&b[i\_0];

b\_tvin\_wrapc\_buffer[hls\_map\_index].range(31, 0) = b\_tmp\_mem.range(31, 0);

hls\_map\_index++;

}

}

}

}

}

// dump tv to file

for (int i = 0; i < 8; i++)

{

sprintf(tvin\_b, "%s\n", (b\_tvin\_wrapc\_buffer[i]).to\_string(SC\_HEX).c\_str());

aesl\_fh.write(AUTOTB\_TVIN\_b, tvin\_b);

}

tcl\_file.set\_num(8, &tcl\_file.b\_depth);

sprintf(tvin\_b, "[[/transaction]] \n");

aesl\_fh.write(AUTOTB\_TVIN\_b, tvin\_b);

// release memory allocation

delete [] b\_tvin\_wrapc\_buffer;

// [[transaction]]

sprintf(tvin\_c, "[[transaction]] %d\n", AESL\_transaction);

aesl\_fh.write(AUTOTB\_TVIN\_c, tvin\_c);

sc\_bv<32>\* c\_tvin\_wrapc\_buffer = new sc\_bv<32>[8];

// RTL Name: c

{

// bitslice(31, 0)

{

int hls\_map\_index = 0;

// celement: c(31, 0)

{

// carray: (0) => (7) @ (1)

for (int i\_0 = 0; i\_0 <= 7; i\_0 += 1)

{

// sub : i\_0

// ori\_name : c[i\_0]

// sub\_1st\_elem : 0

// ori\_name\_1st\_elem : c[0]

// regulate\_c\_name : c

// input\_type\_conversion : \*(int\*)&c[i\_0]

if (&(c[0]) != NULL) // check the null address if the c port is array or others

{

sc\_lv<32> c\_tmp\_mem;

c\_tmp\_mem = \*(int\*)&c[i\_0];

c\_tvin\_wrapc\_buffer[hls\_map\_index].range(31, 0) = c\_tmp\_mem.range(31, 0);

hls\_map\_index++;

}

}

}

}

}

// dump tv to file

for (int i = 0; i < 8; i++)

{

sprintf(tvin\_c, "%s\n", (c\_tvin\_wrapc\_buffer[i]).to\_string(SC\_HEX).c\_str());

aesl\_fh.write(AUTOTB\_TVIN\_c, tvin\_c);

}

tcl\_file.set\_num(8, &tcl\_file.c\_depth);

sprintf(tvin\_c, "[[/transaction]] \n");

aesl\_fh.write(AUTOTB\_TVIN\_c, tvin\_c);

// release memory allocation

delete [] c\_tvin\_wrapc\_buffer;

// [call\_c\_dut] ---------->

CodeState = CALL\_C\_DUT;

float AESL\_return = simple(a, b, c);

CodeState = DUMP\_OUTPUTS;

// [[transaction]]

sprintf(tvout\_c, "[[transaction]] %d\n", AESL\_transaction);

aesl\_fh.write(AUTOTB\_TVOUT\_c, tvout\_c);

sc\_bv<32>\* c\_tvout\_wrapc\_buffer = new sc\_bv<32>[8];

// RTL Name: c

{

// bitslice(31, 0)

{

int hls\_map\_index = 0;

// celement: c(31, 0)

{

// carray: (0) => (7) @ (1)

for (int i\_0 = 0; i\_0 <= 7; i\_0 += 1)

{

// sub : i\_0

// ori\_name : c[i\_0]

// sub\_1st\_elem : 0

// ori\_name\_1st\_elem : c[0]

// regulate\_c\_name : c

// input\_type\_conversion : \*(int\*)&c[i\_0]

if (&(c[0]) != NULL) // check the null address if the c port is array or others

{

sc\_lv<32> c\_tmp\_mem;

c\_tmp\_mem = \*(int\*)&c[i\_0];

c\_tvout\_wrapc\_buffer[hls\_map\_index].range(31, 0) = c\_tmp\_mem.range(31, 0);

hls\_map\_index++;

}

}

}

}

}

// dump tv to file

for (int i = 0; i < 8; i++)

{

sprintf(tvout\_c, "%s\n", (c\_tvout\_wrapc\_buffer[i]).to\_string(SC\_HEX).c\_str());

aesl\_fh.write(AUTOTB\_TVOUT\_c, tvout\_c);

}

tcl\_file.set\_num(8, &tcl\_file.c\_depth);

sprintf(tvout\_c, "[[/transaction]] \n");

aesl\_fh.write(AUTOTB\_TVOUT\_c, tvout\_c);

// release memory allocation

delete [] c\_tvout\_wrapc\_buffer;

// [[transaction]]

sprintf(tvout\_ap\_return, "[[transaction]] %d\n", AESL\_transaction);

aesl\_fh.write(AUTOTB\_TVOUT\_ap\_return, tvout\_ap\_return);

sc\_bv<32> ap\_return\_tvout\_wrapc\_buffer;

// RTL Name: ap\_return

{

// bitslice(31, 0)

{

// celement: return(31, 0)

{

// carray: (0) => (1) @ (0)

{

// sub :

// ori\_name : AESL\_return

// sub\_1st\_elem :

// ori\_name\_1st\_elem : AESL\_return

// regulate\_c\_name : return

// input\_type\_conversion : \*(int\*)&AESL\_return

if (&(AESL\_return) != NULL) // check the null address if the c port is array or others

{

sc\_lv<32> return\_tmp\_mem;

return\_tmp\_mem = \*(int\*)&AESL\_return;

ap\_return\_tvout\_wrapc\_buffer.range(31, 0) = return\_tmp\_mem.range(31, 0);

}

}

}

}

}

// dump tv to file

for (int i = 0; i < 1; i++)

{

sprintf(tvout\_ap\_return, "%s\n", (ap\_return\_tvout\_wrapc\_buffer).to\_string(SC\_HEX).c\_str());

aesl\_fh.write(AUTOTB\_TVOUT\_ap\_return, tvout\_ap\_return);

}

tcl\_file.set\_num(1, &tcl\_file.ap\_return\_depth);

sprintf(tvout\_ap\_return, "[[/transaction]] \n");

aesl\_fh.write(AUTOTB\_TVOUT\_ap\_return, tvout\_ap\_return);

CodeState = DELETE\_CHAR\_BUFFERS;

// release memory allocation: "a"

delete [] tvin\_a;

// release memory allocation: "b"

delete [] tvin\_b;

// release memory allocation: "c"

delete [] tvout\_c;

delete [] tvin\_c;

// release memory allocation: "ap\_return"

delete [] tvout\_ap\_return;

AESL\_transaction++;

tcl\_file.set\_num(AESL\_transaction , &tcl\_file.trans\_num);

return AESL\_return;

}

}