fxpemu user manual

Title	fxpemu (Fixed-point emulation library for ANSI C)
Author	Nikolaos Kavvadias (C) 2010, 2011, 2012, 2013, 2014
Contact	nikos@nkavvadias.com
Website	http://www.nkavvadias.com
Release Date	05 October 2014
Version	0.1.0
Rev. history	
v0.1.0	2014-10-05 Updated for first public release on Github.
v0.0.4	2013-04-06 Standalone version.
v0.0.3	2011-04-27 Changes to fxp_to_int32. Added fxp_ceil.
v0.0.2	2010-11-24 Added fxp_to_double.
v0.0.1	2010-11-23 Added fxp, fxp_resize.
v0.0.0	2010-11-22 Initial version. Includes the definitions, declarations and implementations of: • BITASSIGN macro plus ABS, MAX, MIN, BITGET • QuantizationType, OverflowType enums • fixed struct • fxp_quantize, fxp_overflow • fxp_create, FIXED_INIT macro, INIT macro • fxp_add, fxp_sub, fxp_mul, fxp_div, int32_to_fxp, fxp_to_int32, fxp_neg, fxp_abs, fxp_min, fxp_max, fxp_to_string, fxp_print, string_to_fxp, double_to_fxp

1. Introduction

`fxpemu` is an ANSI C library that implements basic fixed-point arithmetic data structures alongside a usable side. Its purpose is to be used in the context of software and hardware compilers and code generators.

The implementation of the fixed struct, the "overflow" and "rounding" routines, and the general form of `fixed_xyz` routines is based on the following reference:

Stephen A. Edwards,
"Using program specialization to speed SystemC fixed-point simulation,"
Proc. of the Workshop on Partial Evaluation and Program Manipulation (PEPM 2006), pp. 21-28, Charleston, South Carolina, USA, January 2006.

The draft of the reference paper is available (as of 2014-Sep-20) from:

- http://www.cs.columbia.edu/~sedwards/papers/edwards2006using.pdf

It should be noted that the additional functionality is based on newly-designed code.

2. File listing

The `fxpemu` abstract data types and API code base includes the following files:

/fxpemu	Top-level directory
AUTHORS	List of authors.
LICENSE	License argeement (Modified BSD license).
Makefile	GNU Makefile for building `test-fxpemu.exe`.
README.rst	This file.
README.html	HTML version of README.
README.pdf	PDF version of README.
VERSION	Current version.
fxpemu.c	C code implementing the fixed-point arithmetic API.
fxpemu.h	C header file for the above. Also defines data structures along some arithmetic macros needed.
rst2docs.sh	Bash script for generating the HTML and PDF versions.
test-fxpemu.c	Application code for exercising basic functionality of the implemented fixed-point arithmetic API.

3. Function reference

This section provides a quick reference of the functions used for implementing the `fxpemu` data structures (enums, structs and basic macros) and API.

3.1 Data structures

3.1.1 `QuantizationType` enum

This enum provides definitions for the possible quantization (truncation or rounding) modes in fixed-point arithmetic. It is defined as follows:

```
typedef enum {
  UNKNOWN_QUANT_TYPE = -1,
  AC_TRN,         /* Default in ACDT (Algorithmic C Datatypes). */
  AC_TRN_ZERO,
  AC_RND,         /* Default in VHDL fixed-point package. */
  AC_RND_ZERO,
  AC_RND_INF,     /* Implemented as "SC_RND" in SystemC 2005. */
  AC_RND_MIN_INF,
 AC_RND_CONV
} QuantizationType;
```

3.1.2 `OverflowType` enum

This enum provides definitions for the possible overflow-handling (wrapping or saturation) modes in fixed-point arithmetic. It is defined as follows:

```
typedef enum {
  UNKNOWN_OVERFLOW_TYPE = -1,
  AC_WRAP,        /* Default in ACDT (Algorithmic C Datatypes). */
  AC_SAT,         /* Default in VHDL fixed-point package. */
  AC_SAT_ZERO,
  AC_SAT_SYM
} OverflowType;
```

3.1.3 `fixed` struct

This struct defines the `fixed` data structure which essentially is the container of a fixed-point value. It consists of the `val` (value), ``wl` (word length), `iwl` (integer word length), `lbp` (location of binary point), and the `overflow` and `rounding` flags. The `fixed` struct is defined as follows:

```
typedef struct {
  int val;      /* 32-bit value value of the number */
  int wl;       /* Word length, in bits */
  int iwl;      /* Integer word length, in bits */
  int lbp;      /* Location of binary point, in bits */
  int overflow;
  int rounding;
} _fixed;
typedef _fixed fixed;
```

3.1.4 `FIXED_INIT` macro

The `FIXED_INIT` macro initializes a fixed-point variable `x` to a set of given values: `w, i, l, ovr, rnd` according to the definition of the `fixed` struct. It is defined as follows:

```
#define FIXED_INIT(x, w, i, l, ovr, rnd) \
  x.wl = w; \
  x.iwl = i; \
  x.lbp = l; \
  x.overflow = ovr; \
  x.rounding = rnd
```

3.2 API

3.2.1 fxp_quantize

```
void fxp_quantize(fixed *r);
```

Apply fixed-point arithmetic quantization rules to fixed r. These rules are used for handling the low-significance bits of r.

3.2.2 fxp_overflow

```
void fxp_overflow(fixed *r);
```

Apply fixed-point arithmetic overflow rules to fixed r. These rules are used for handling the high-significance bits of r.

3.2.3 fxp_create

```
fxp_create(int val, int iwl, int fwl,
" OverflowType ovr_mode, QuantizationType rnd_mode,"
" int offset);"
```

Constructor for a signed fixed-point variable.

3.2.4 fxp

```
fixed fxp(int value, int left, int right, char is_signed,
char rounding);
```

Constructor for a signed fixed-point variable. This version is provided for compatibility to a third-party tool/plugin, namely the Agility RMS.

This constructor cannot set the overflow mode. The overflow mode of AC_SAT is used by default, and can be changed by explicit modification of the "overflow" field of a fixed-point variable. An "offset" (for establishing.

"is_signed" is currently left unused.

3.2.5 fxp_resize

```
void fxp_resize(fixed *r, int L, int R, char sign);
```

Extends or shrinks the number of bits left or right of the binary of a specified "fixed" arithmetic type. This version uses an additional specifier, "sign" which can take the values of 'u' or 's'. For proper use, the specified sign must be the same to that supposed for fixed r.

The location of the binary-point (LBP(.)) is not affected.

3.2.6 `fxp_add`

```
void fxp_add(fixed *r, fixed *a, fixed *b);
```

Fixed-point addition.

3.2.7 `fxp_sub`

```
void fxp_sub(fixed *r, fixed *a, fixed *b);
```

Fixed-point subtraction.

3.2.8 `fxp_mul`

```
void fxp_mul(fixed *r, fixed *a, fixed *b);
```

Fixed-point multiplication.

3.2.9 `fxp_div`

```
void fxp_div(fixed *r, fixed *a, fixed *b);
```

Fixed-point division.

3.2.10 `int32_to_fxp`

```
fixed int32_to_fxp(int a, int w, int i, int lbp, int ovr,
int rnd);
```

Convert a signed int to a fixed-point value.

3.2.11 `fxp_to_int32`

```
int fxp_to_int32(fixed *a);
```

Convert a fixed-point value to a signed int.

3.2.12 `fxp_neg`

```
void fxp_neg(fixed *r, fixed *a);
```

Fixed-point negation.

3.2.13 `fxp_abs`

```
void fxp_abs(fixed *r, fixed *a);
```

Fixed-point absolute value.

3.2.14 `fxp_min`

```
void fxp_min(fixed *r, fixed *a, fixed *b);
```

Fixed-point minimum of two numbers.

3.2.15 `fxp_max`

```
void fxp_max(fixed *r, fixed *a, fixed *b);
```

Fixed-point maximum of two numbers.

3.2.16 `fxp_to_string`

```
void fxp_to_string(fixed *a, int b, char *s);
```

Converts a fixed-point number of the form i.f to a string, assuming arithmetic in base b.

3.2.17 `fxp_print`

```
void fxp_print(fixed *a, int b);
```

Prints the argument as a fixed point i.f number in base b.

3.2.18 `string_to_fxp`

```
void string_to_fxp(char *s, int b, fixed *r);
```

Reads a string representing a b-base (b=2 for binary) fixed-point number and performs the conversion to an actual "fixed"-type number.

3.2.19 `double_to_fxp`

```
void double_to_fxp(double d, fixed *a);
```

Converts a double (64-bit floating-point) to a binary fixed-point number with possible loss of precision.

3.2.20 `fxp_to_double`

```
double fxp_to_double(fixed *a);
```

Converts a binary fixed-point number (actually its integer emulation) to the corresponding double (64-bit floating-point) representation.

3.2.21 `fxp_ceil`

```
void fxp_ceil(fixed *r, fixed *a);
```

Fixed-point ceiling (rounding to positive infinity).

4. Usage

The implementation of the fixed-point arithmetic API can be used in context of a provided test application, named `test-fxpemu.c`. The Makefile can be used for building this application as follows:

```
$ cd fxpemu
$ make clean ; make
```

This will also build the static library implementation of fxpemu, which is the `libfxpemu.a` file. Third-party/user applications can be implemented by including the `fxpemu.h` header file and statically linking to the library.

To run the test application do the following:

```
$ ./test-fxpemu.exe
```

Executing the application will produce a stream of diagnostic messages to standard output.

5. Prerequisites

- Standard UNIX-based tools (tested with gcc-4.6.2 on MinGW/x86 and gcc-4.8.2 on Cygwin/x86/Windows 7)
  - make

  On Windows (e.g. Windows 7, 64-bit), MinGW (http://www.mingw.org) or Cygwin (http://sources.redhat.com/cygwin) are suggested.

  The sources should be able to compile without any messages on any recent Linux distribution.