在C_C 算法设计中使用任意位宽 PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/245/2021_2022__E5_9C_A8C_C___E7_AE_c97_245975.htm 开发定点(fixed-point)算法时，通常需要在设计功能性、数字精度建模、及验证(仿真)速度之间取得一个平衡。现在，一种新的数据类可使此过程简单化，由此得到更简单精确的建模精度、更好的数字求精、及更快的验证周期，而ANSI C/C 正是开发这种数字求精算法的最佳语言。 某此算法天生就适用于操作整数，或那些理想中的实数(如数字滤波器的系数)，它们也可能会使用浮点或定点类型。一般而言，在算法开发的早期阶段，会经常用到C语言的float或double浮点类型，因为它们可提供一个非常大的动态数据范围，且对大多数程序来说都是适用的。 使用C内置的float类型来建模一个FIR滤波器 算法可进行数字求精，以便使用定点算术来降低最终硬件或软件实现的复杂性。在硬件方面，将整数或定点算术限制为最小位宽，可在本质上满足性能、空间、能耗的需要.如果实现中用到了DSP处理器，那么把算法限制为整数或定点算术，就可为特定程序使用尽可能便宜的处理器。 定点算术的建模可通过C语言内置的浮点或整数类型来完成，这做的话，需要显式编码并受限于C中浮点数及整数可表示的最大数：64位整数或53位尾数.这些都会给操作数的位宽带来更多的限制，例如，2个33位的数相乘，会超过64位C整数可表示的范围。图2演示了一个FIR滤波器的例子，但temp变量限制为15位的定点精度，其中10位用于整数位。在这个实现中，LSB的右部位被舍弃(量化模型的截断)，而MSB的左部位也被舍弃(包装的溢出模型)，应该意

识到，使用float(或double)的模型在精度上是受限的，且不能再次合成(synthesis)。同样，由于有取整模型的严格位精度定义有先，又由于内置浮点类型的取整将会先被应用，所以对除法这样的操作来说，就非常难实现了。 使用float建模定点行为 当许多算法都能依赖本地C数据类型的精度来编写时，对支持任意长度的整数及定点算法，大家就会抱有极大的期望，而硬件描述语言(HDL)如VHDL，走的也是同一条路。随着C/C 越来越多地被用于高级合成与验证工具(High-Level Synthesis and Verification tools)，也证明了这种语言本质上有一个足以满足当前及未来程序需要的数据类型库。任意长度类型的支持，也可使数据类型的行为有一个统一的定义，而统一的语义则避免了人工实现上的一些限制。 算法C数据类型 算法C数据类型是一种基于类的C 库，其实现了任意长度的整数及定点类型，而这些可自由访问的类型有一系列好处，包括统一及良好定义的语义，还有媲美C/C 内置数据类型的运行时速度，对比SystemC中相应的类型，其运行速度也超过10倍以上。这些数据类型能用于任何符合C 或SystemC规范标准的程序中，并拥有高度可合成的语义。 语义 语义的统一性与一致性是避免在算法中，发生功能性错误的关键，以下的例子，也说明了这点： 众所周知，变量ActLength的范围为1至255，万一编译器的合成不知道其范围，就不能进行相应的优化，它的声明就会从int变为更严格的sc_uint类型.虽然合成会得到更好的结果，但设计就仿真得不正确了。在经过一番调试之后，找到了问题的源头：在比较表达式k >= ActLength中，两个操作数变成了一个signed int与一个unsigned long long(为64位无符号整数，其是sc_uint类型的基类型)之间的比较。

对此的解释是：C/C 整数提升规则指定了在进行比较之前，会把操作数int提升为一个unsigned long long，例如，如果k的值为 -1，在提升为unsigned long long之后，它会变成2^64 1。像这样语义中的问题一般会非常难以察觉，且是与位宽相关的，例如，可能有人想扩大某个现有算法的位宽，只有看到结果时，才知道是行不通的.这个问题也可能是与特定平台相关的，例如，对1 算法C数据类型被设计用于提供统一且一致的语义，因此，它们是可预测的，例如，对有符号数混用一个无符号操作数仍会产生期望的结果.这些类型的长度不受限制，所以就不存在所谓的精度问题。所有的操作包括移位和除法都有完整且一致性的定义，混合不同的类型也能得到期望的结果，如，当x为一个C内置类型，而y是一个算法C类型时，表达式x y和y x均能返回相同的结果。 运行时间 我们的目的是为了在支持任意长度类型及避免用户碰到前述语义问题的前提下，得到使用内置类型(位宽不超过64位)手工C/C 编码优化过的运行时间。算法C类型是为快速执行及易于合成的语义而设计及实现的，所有操作的位宽由C 编译器静态确定，这就避免了动态内存分配，减少了运行时间，也使得语义更加易于合成。另外，实现也为速度进行了优化，因此可能会调用更多的专用及高效代码，充分利用了当今编译器的优化特性。 表1：规格化为ac_fixed的运行时间比较 表1是当定点算术用算法C定点类型ac_fixed来建模时，各种不同的运行时比较.float的实现在图2中，sc_fixed_fast为SystemC中精度受限的定点数据类型，sc_fixed为任意精度的定点类型。实际中对FIR滤波器进行10^8次调用，TRN/WRAP的运行时间为6.5秒，RND/SAT为29秒。其他类型的运行时间也能从这张表中

依次推出，如sc_fixed TRN/WRAP将花费6.5s × 227 = 1476s(将近25分钟)。作为参考，图1中的算法(使用无定点建模的float)花费时间为3.5s(比起使用定点建模的ac_fixed，慢了近两倍)。上述的运行时间数据，均由GCC 4.1.1测量得来，而在之前版本的GCC或Visual C 2005中得到的数据大致接近。 另外，运行时间也能通过整型数据类型或位操作进一步缩短。表2为一个DCT算法的相应结果，它由一个每次读写2位的移位操作得来，与此对比的运行时间为SystemC精度受限的sc_int与任意长度的sc_bigint。 表2：规格化为ac_int的运行时间比较 结论 基于通用标准ANSI C，这种新的整数与定点算法C类型允许算法及系统设计者指定任意位宽，从而提供比传统数据类型高200倍的仿真效率。这些新数据类型可成为C-to-RTL设计链中非常有价值的一环，及在整个实现流程中保证了任意位宽的精度。 