信号处理算法设计流程

以编写基2-FFT的C代码为例

用绳量给我的地界

坐落在佳美之处

我的产业实在美好

杜伟韬

duweitao@cuc.edu.cn

2013年 6月 2日于 北京定福庄

献给广播学院的核桃林，还有我的老师们

目 录

0 引言

数字信号处理系统的应用需求和实现形式具有非常多样化的特征，在信号处理领域，存在一种分类习惯，即将信号处理系统分为“信号分析”与“信号合成”两类功能，在不同的系统中，两者的使用顺序也不尽相同，例如：在一个雷达站或是B超成像仪系统中，首先是发射设计好的信号波形，然后通过感知回波信号特征参数来估计被测物特征；类似还有数字无线通信系统，发射部分将用户输入的比特序列根据通信协议合成为发射波形，通过信道后再被接收机进行分析还原得到用户比特序列。与此相反，在语音和音视频等多媒体应用中，输入的样值序列通常首先被编码器进行分析并转化为参数码流，在通过二进制离散信道（磁盘或是某种通信协议的数据链路层）后被解码器合成并还原为多媒体信息。

无论是信号分析或是合成系统，对信号进行离散傅里叶变换和数字滤波均是最基本的操作，本节首先以基2-FFT为例，展示信号处理算法是如何从推导数学公式向处理器上运行的算法代码的映射过程，另外由于定点处理器的广泛应用，接下来的内容对算法数制系统做出了一般性的介绍，最后，本节末尾部分给出了两种常用的信号处理算法的验证方法与流程。

1 信号处理算法设计流程

数字信号处理算法的源头是数学，各种分析与处理算法均使用符号体系与公式进行描述。信号处理算法设计是一个递进过程，首先是在应用背景理论体系下的算法原型，然后映射成可计算的近似公式，例如离散傅里叶变换DFT之于离散时间傅里叶变换DTFT，如果运算量过于庞大，则还需设计快速算法，例如FFT之于DFT。

随着计算机工具的发展，使用数学公式所表达的算法原型可以用高级编程语言进行仿真验证。复杂系统通常在数据流层面仿真，底层基础算法通常在向量层面进行验证，另外，由于C语言在EDA行业所得到的广泛支持，其目前仍有拥有其他工具所无法比拟的跨平台兼容性，因此和向量编程语言相比，使用C语言对算法原型进行描述同样具有重要意义。本节后续部分以基-2 FFT 算法为例，阐述将算法从数学公式映射算法编程模型过程中需要考虑的问题。

1.1 算法原型与强度缩减

算法原型是指在理论模型体系中定义出的数学表达式，这些表达式通常含有连续变量因此无法进行数值计算，比如建立在归一化角频率域的DTFT，其$X(\omega)=\sum_{n=-\infty}^{+\infty}x(n)\cdot e^{-j\omega n}$需要在无穷时序尺度计算得到连续角频率域的结果函数。不可计算的理论模型通常会有等价的近似计算模型，例如DFT $X(k)=\sum_{n=0}^{N-1}x(n)\cdot e^{-j\frac{2\pi}{N}nk}$ 便是DTFT函数的等间隔抽样序列。强度缩减是指对于一个高计算强度的计算过程，通过某种分解方法将其映射为若干个过程类似但计算强度较小的子过程，再通过某种方法把子过程的结果合成为最终总的结果。例如在基-2频率抽取（DIF：Decimation-in-Frequency）的FFT算法中，式(1)所描述的DFT定义式计算过程，可以分解为式(2)的基-2频率抽取过程。

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot W_N^{nk}, \quad n = 0, 1, ..., N-1 \tag{1}$$

$$\begin{cases} X(2r) & = & \sum\limits_{n=0}^{\frac{N}{2}-1} \left[x(n) + x(n+\frac{N}{2})\right] \cdot W_{N/2}^{nr}, & r = 0, 1, ...\frac{N}{2} \\ X(2r+1) & = & \sum\limits_{n=0}^{\frac{N}{2}-1} \left[x(n) - x(n+\frac{N}{2})\right] \cdot W_N^n \cdot W_{N/2}^{nr}, \end{cases} \tag{2}$$

由此可见，在采用基-2 DIF 分解的FFT计算过程中，一个计算强度为N的DFT过程，可以由基-2 DIF分解为两个强度N/2的DFT过程，同时该分解方法需要消耗N次加法和N/2次乘法，重复使用该方法则会得到最终的FFT计算过程。以上实例代表了设计快速算法的典型过程，即：将高强度计算过程分解为若干具有同样或类似形式的低强度过程；递归使用该分解方法直至将子过程计算强度降至最低。同时，分解方法所带来的计算开销必须小于其产生的子过程，否则使用该方法产生的最终计算过程的运算开销会高于按照定义式直接计算。

1.2 循环设计与存储映射

1.2.1 循环嵌套模型

仍以基-2 DIF FFT为例，对于快速算法的理论研究工作，当得到式(2)的分解表达式并评估运算量之后，其主体工作已经完成，但对于算法的实现者而言，仅有递推关系不足以将其直接映射为实现代码，需要对算法过程进行更加深入描述。

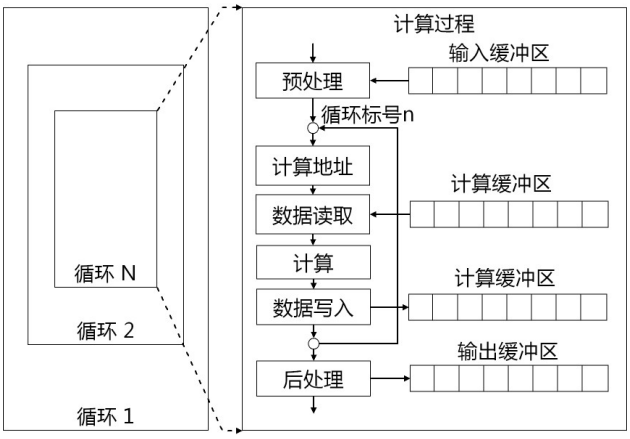

图 1: 信号处理算法嵌套执行过程

信号处理算法仿真执行过程通常包括：*1*) 初始化过程：用于向系统申请内存以及计算出运行时需要使用的常量系数。*2*) 计算过程：从输入缓冲区获取数据；利用内部缓冲区计算数据；向外部缓冲区输出计算结果。*3*) 结束过程：释放申请的系统内存。

信号处理算法的实现过程通常由若干层循环嵌套构成，图1 绘出了循环算法嵌套的执行过程，其中，从

外层循环视角，可将内层循环视为一简单的执行过程，而从循环内部视角可以将执行过程分为：*1*) 预处理阶段：从循环的输入缓冲区读取数据至计算缓冲区，同时执行必要的预处理计算操作。*2*) 主循环体阶段：以标号 n 进行循环计数; 根据各循环标号计算各种读写地址; 读取缓存数据；计算数据并更新缓存。*3*) 后处理阶段：执行必要的后处理计算操作并将计算结果送至循环的输出缓冲区。

1.2.2 循环标号与变量寻址

对于基-2 频率抽取(DIF)的FFT，图2 绘出了一个8点基-2分解FFT的蝶形图，图中描述的全部计算过程需要用循环变量表达式的方式进行描述，通过研究该图可以发现，基-2 FFT的计算过程概念上乃是三层循环结构。最外层循环称之为“级（LEVEL）”，对于N点基-2 FFT，级数为$\log_2^N$，究其原因每进行一次基-2分解就会产生一级（Level）。第二层循环称之为“组（Group）”，每一级由若干组构成，因为每次基-2分解产生的短DFT并不能直接计算，还需再进行递归分解直至生成2点DFT方可。通过数学推导和观察图形可知，在不同级中，组数目N_G是级标号i_L的函数：$N_G(i_L) = 2^{i_L}$。第三层循环称之为“蝶形（Butterfly）组中蝶形计算的数目N_B亦为级标号i_L的函数$N_B(i_L) = N/2^{i_L}$。

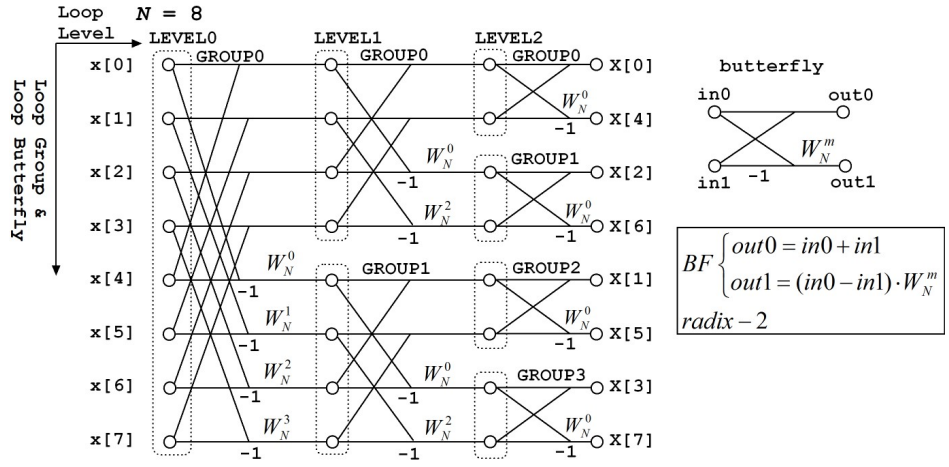

图 2: 8点FFT基-2 DIF分解蝶形图

再从数据存储的视角观察图2，该算法的输入数据需要占用一列（N点）存储器，按照级（Level）的顺序对数据进行运算，每级运算的结果仍然需要使用一列（N点）存储器保存，另外每一级的旋转因子W_N^i也需要占用N/2点存储器。在实现该算法时，对于设计者而言，需要制定数据的读写策略将蝶形图中的所有计算全部遍历，如前文所述可以按照“级-组-蝶形”的循环嵌套顺序，以蝶形运算为单位对数据进行访问，该策略中循环变量之间依赖关系如表 1所示:

则在“级-组-蝶形”的三层循环中，蝶形计算的两个输入数据地址BA_0，BA_1 均为三层循环标号的函数，其中，$BA_0(i_{BG}, i_G, i_L) = i_{BG} + i_G + 2 \cdot N_{BG}$，$BA_1(i_{BG}, i_G, i_L) = BA_0 + N_{BG}$，同样该次蝶形运算所对应的旋转因子$W_N^m$亦为循环标号的函数，$m(i_{BG}, i_G, i_L) = i_{BG} \cdot 2^{i_L}$。

表 1: FFT算法循环变量依赖关系

变量名称	含义	数值或范围
N	FFT点数	2的整数次幂
N_L	Level 个数	$\log_2^N$
N_G	Level中Group个数	2^{i_L}
N_{BL}	Level中蝶形个数	$2^{(N_L-1)}$
N_{BG}	Group中蝶形个数	$2^{(N_L-i_L-1)}$
i_L	Level的标号	$0 \sim N_L-1$
i_G	Level中Group标号	$0 \sim N_G-1$
i_{BL}	Level中蝶形标号	$0 \sim N_{BL}-1$
i_{BG}	Group中蝶形标号	$0 \sim N_{BG}-1$

1.2.3 常量系数的存储与动态计算

除规划待处理数据的存储方案之外，计算使用的常量系数的存储策略也要制定，在基-2 FFT的计算过程中，需要使用的旋转因子为$W_N^0 \sim W_N^{N/2-1}$ 共计 N/2 点复数数据，共计N点实数数据。在以上N/2点的复数数据中，其实部为半周期余弦波形，虚部为半周期正弦波形，存在对称性，可利用此种对称性减少系统存储开销，此种策略的额外代价是动态计算数据读取地址的额外计算开销。图3 给出了在N=16情况下，旋转因子虚部一个周期波形的样值序列。

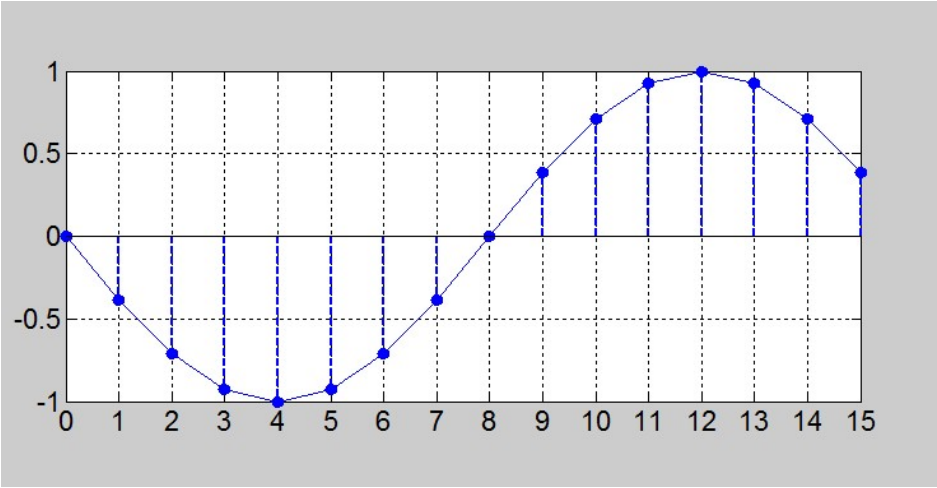

图 3: 16点FFT旋转因子虚部样值序列

式(3)给出了旋转因子的定义式，若仅保存虚部的前半部分N/4周期波形，则旋转因子实部的计算过程如式(4)所示，虚部的计算过程如式(5)所示。

$$W_N(r) = \cos\frac{2\pi r}{N} - j\sin\frac{2\pi r}{N}, r = 0 \sim \frac{N}{2} - 1 \tag{3}$$

$$\mathrm{Re}[W_N(r)] = \begin{cases} -\mathrm{Im}[W_N(\frac{N}{4} - r)], r = 0 \sim \frac{N}{4} - 1 \\ \mathrm{Im}[W_N(r)_{\bmod \frac{N}{4}}], r = \frac{N}{4} \sim \frac{N}{2} - 1 \end{cases} \tag{4}$$

$$\mathrm{Im}[W_N(r)] = \begin{cases} \mathrm{Im}[W_N(r)], r = 0 \sim \frac{N}{4} - 1 \\ -1, r = \frac{N}{4} \\ \mathrm{Im}[W_N(\frac{N}{2} - r)], r = \frac{N}{4} \sim \frac{N}{2} - 1 \end{cases} \tag{5}$$

从式(4) (5) 中可以看到，在利用对称性计算旋转因子的访问地址过程中，存在判断标号数值范围的计算，因此带来破坏处理器指令流水线的可能，具体情况取决于目标处理器结构对条件执行和指令预读取功能的支持情况，这亦是压缩存储所带来的代价。

除计算蝶形之外，位反序寻址策略也需进行设计，本例所采用的基-2 DIF分解策略对应“正序输入、基-2分解、位反序输出”的寻址过程，对于标号数据的位反序计算，可以通过位运算或是查表的方式进行，位运算的方式适合使用电路逻辑实现，在处理器上实现或消耗较多的指令周期，查表方式适合在FFT点数较小的情况下用处理器实现，当在处理器上进行较大点数的FFT计算时，可以使用两者结合的方法。例如建立256点的位反序查找表并辅以少量位运算，可以对8比特及以下的标号数据进行位反序结果查找。当标号字长大于8比特时，可以通过将标号数据分解为多个字节，每字节分别查表并将查表结果反序排列并辅以少量位运算得到最终的标号位反序结果。同样，该过程也体现了“计算-存储”的折中策略。

1.3 数制系统

二进制的数字电路只能表达0、1信息，信号处理算法通常使用复数或实数域，使用二进制来承载算法的数据变量以及它们之间的运算规则称之为数制系统。在算法原型设计过程以及使用仿真工具建模验证算法的先期阶段，通常以浮点格式保存数据。尽管若干型号的嵌入式处理器和FPGA芯片具备硬件浮点计算的能力，但整数的乘加运算仍是信号处理硬件平台的必备能力，由此产生了多种使用整数运算来实现数学计算的方案，列举如下。

1.3.1 整数格式

整数是最基本的数据格式，常用格式为：*1*) 无符号数 *2*) 有符号原码 *3*) 有符号二进制补码。表2 给出了4比特2进制数据在不同数制系统下所代表的数值。

二进制补码被广泛用于计算机编程语言以及硬件电路描述语言中的有符号整数的格式，这是因为其具有以下优点：*1*) 消除了+0 和 -0的歧义; *2*) 把减法操作映射为“取相反数-相加”的统一操作; *3*) 良好的溢出特性，即，如果多个二补码数相加时，中间结果发生溢出，若数学上的最终结果不溢出则实际运算的最终结果也不溢出。

另外，需注意补码乘法器的扩展符号位，即，使用处理器提供的补码乘法指令或是EDA工具提供的补码乘法器进行运算时，最终结果数据的最高2比特总是同样数值，且为符号位，究其原因是补码乘法器电路结构所致，该项细节在不同工具链中可能存在差异，最好实践验证。

除乘、加运算之外，补码数据存在符号扩展问题，即，不同字长的补码数据在进行加法时，需两者字长一致，则要将短字长数据高位补齐，并以其符号位填充内容。由此，补码数据的右移也存在两种运算，即以0填充空白位的逻辑右移，以及用符号位填充空白位的算术右移。

表 2: 4比特二进制数的不同格式

二进制数	无符号数	有符号原码	二进制补码
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	8	0	-8
1001	9	-1	-7
1010	10	-2	-6
1011	11	-3	-5
1100	12	-4	-4
1101	13	-5	-3
1110	14	-6	-2
1111	15	-7	-1

1.3.2 定点格式

定点格式是使用整数运算进行有理数计算的最基本格式，如图4所示，参与计算的整数字长为WL，被分为符号位S，整数I，小数F 共三部分，其中符号位为1比特，整数部分字长IWL，小数部分字长FWL。制定定点格式中各个部分的字长时，需要首先分析算法的输入数据、中间结果数据、输出数据的精度和动态范围需求。

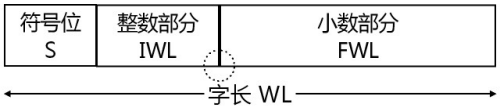

图 4: 定点数据格式

同种定点格式数据进行加法时，由于操作数的小数点已经对齐，则不需单独处理整数部分和小数部分，直接进行整数加法运算即可。不同格式的定点数在进行加法之前，需通过符号扩展及算术右移的操作将两者字长调整一致且小数点对齐后再相加。和整数加法类似，定点数加法存在着动态范围增加的问题，即，两个二进制定点数相加后，整数部分字长可能会增加1个比特，为避免潜在的溢出危险，通常有三种策略：*1*) 对于字长较长的处理器，使用足够字长来表达整数部分，确保不会溢出。*2*) 对于字长较短的处理器，在加法操作之前，首先将两操作数算术右移。*3*) 使用数字电路实现算法时，可以为加法的输入和输出定义不同的整数字长。图5 绘出了一种全精度定点加法的字长配置策略。

与定点加法的字长扩展方式不同，定点乘法的字长扩展是以小数点为中心，向两端扩展，图6绘出了全精度定点乘法的字长扩展情况，注意其中扩展符号位S1 总是和符号位S0 保持相同数值，需要说明的是，实际应用中一般不会使用到高于输入数据的小数精度，所以通常对乘法结果的小数位数要求“等精度”，即和输入精度相等，常用策略有两种：*1*) 对于处理器编程，在乘法操作之前，首先将两操作数算术右移，使得乘法结果的小数精度满足“等精度”需求。*2*) 对于数字电路，先进行全精度乘法，再对不需要保留的小

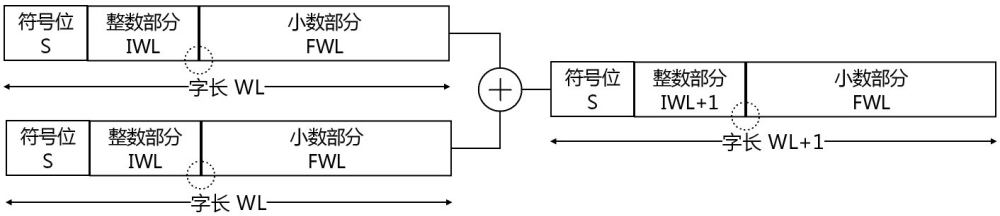

图 5: 定点数据全精度加法

数位数据进行舍入操作。需要格外注意的是，定点乘法结果整数部分字长是两个乘数整数字长之和，这导致乘法操作的整数字长增长速度远高于加法，所以在进行定点乘法之前需要根据应用场景仔细制定数据的字长策略。

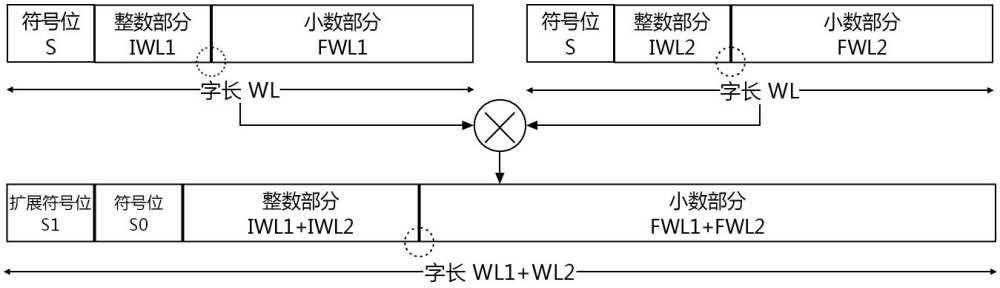

图 6: 定点数据全精度乘法

1.3.3 浮点格式

在科学计算中，浮点是最常用的数据格式，因为其能够同时兼顾数据的动态范围和精度，计算机工业中的浮点数据格式由IEEE754标准进行定义，除定义数据格式之外，为避免计算错误带来的不可预估的后果，该标准对数据计算的各种行为和例外情况也做出了严格的约束。

和通用的标准化浮点格式不同，在嵌入式计算环境中，算法设计者对数据的精度和动态范围可以有一定的预估计，因此，为了同时满足数据动态范围和精度需求，可以采用自定义的浮点格式，主要有用于数字电路的短浮点格式和用于定点处理器的块浮点格式。

所谓短浮点格式是指采用小于标准单精度浮点（32比特）的字长，自行定义底数和阶码长度，这可以降低浮点运算单元的电路尺寸从而可以大量例化该计算单元，对于自定制短浮点格式，其数据格式为符号位S 字长1比特，（纯）小数部分F（底数） 字长FL位，指数E（阶码） 字长EL，其数值为$(2 \cdot S + 1) \cdot F \cdot 2^E$，由于嵌入式计算通常不需高于输入数据的小数精度，因此指数E可以采用无符号整数格式，则EL比特的指数字长可以表达的动态范围为$(-2^{(2^{EL}-1)}, +2^{(2^{EL}-1)})$。和标准浮点数类似，定制浮点的乘法运算为小数部分相乘，指数部分相加，在预先分析算法理论动态范围的前提下，使用合适的指数字长可以确保乘法结果不会溢出，则可不必设计额外的溢出保护电路。定制浮点数的相加运算也需先将操作数的阶码对齐，即将阶码较小数据的底数算术右移，对齐阶码后再相加，需要注意的是，如果底数加法结果发生进位，则需再次调整阶码使得底数重新规格化为纯小数。

所谓块浮点格式是指，将具有相同指数，而尾数不同的一组数据作为数据块，具体实现时，使用整数数组表示各个底数（纯小数），使用单个整数变量表示指数阶码，使用块浮点的复杂之处在于：在计算过程中逐个生成块浮点的各个底数分量时，需动态增加阶码以保证数据块中被计算出的所有底数均为纯小数，即如果当前运算得出的底数包含了整数部分，则需将阶码增加，由于阶码对整个数据块有效，一旦修改阶码，除当前底数外，事先已经计算出的所有底数也需要修正，这会带来较多读写动作从而导致计算效率降低。因此实际应用中不会立即调整阶码和重新修正全部已经计算出的底数分量，而是采用以下策略：*1*) 除设置数据块阶码变量外，额外设置阶码增量变量。*2*) 为底数预留足够的整数部分字长。*3*) 首次生成（初始化或输入数据）块浮点数据时，保证底数分量均为纯小数，阶码增量为0。*4*) 计算出当前底数分量后，判断其整数部分数值，结合当前阶码增量值计算更新阶码增量。*5*) 再次读取底数数据时，首先根据阶码增量值，将底数算术右移至纯小数，再使用其进行计算。

由此可见，块浮点的数据更新过程较为复杂，任一底数的计算过程均和整个数据块数值相关，计算过程调试较为繁琐，因此该格式通常使用处理器软件编程实现。

1.4 算法设计工具与验证流程

信号处理算法的输入数据形式通常分为两种，基于样点的数据流和基于帧的数据块，其代表算法分别是FIR滤波器和FFT运算，经过若干年的技术演进，算法验证工具则从早期基于数据样点进行计算的Fortran、C等编程语言，发展到以Matlab为代表的基于向量的计算模型，以及后来在数字通信领域以Matlab Simulink 和Aglient SystemVue 工具为代表、用于复杂系统仿真的基于数据流的计算模型。

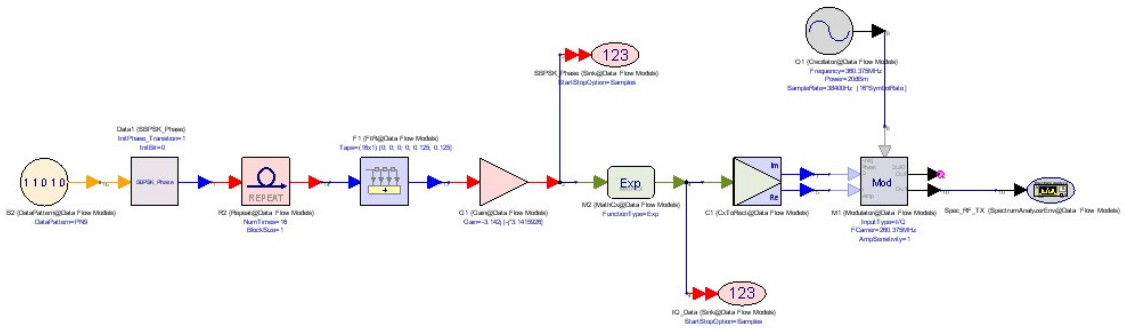

图 7: SystemVue仿真主界面

信号处理算法各种软件化的实现方式中，以C语言为代表的样点级算法描述模型具有最高的硬件执行效率，同时由于其需要人工干预内存的申请与释放，以及编译器具有较强的数据类型检查约束，导致其算法建模的过程中需要关注较多的软件底层细节，致使开发效率较低，因此通常用于在嵌入式环境中作为算法实现的编程语言。以Matlab为代表的向量式的算法描述模型，能够有效的与现代数学计算中的矩阵模型对应，同时由于Matlab能够兼顾样点层面的算法描述，因此向量式的算法描述方式通常用于局部算法模块的建模与验证，比如某种FFT或自适应滤波器算法的建模，并且此种算法模型也有助于参照其实现可用于嵌入式环境的样点级C算法模型。基于同步数据流的算法仿真工具是更为先进的仿真手段，此种仿真工具的广

泛应用得益于以下原因:

1) 在实际系统中，信号在数据转换器的边界处以数据流样点的形式存在，而在中心处理节点经常以各种尺寸数据帧的形式进行处理。*2*) 此种仿真方式能够同时支持样点数据流与向量式的算法建模，并且可以在两种数据处理模式中相互切换。*3*) 如图7所示，拖拽-拼接式的操作辅以完备的模型组件库对于快速构建系统级算法原型具有较高的设计效率。

综上所述，对于以软件方式实现信号处理算法，C语言、Matlab向量编程、同步数据流建模三种方式分别适用于不同层次的设计需求，合理利用三种类型软件工具可以对信号处理系统的算法研发工作进行有力的支撑。

1.4.1 分析验证和对比验证

数字信号处理算法的主要验证手段有两种，分析验证和对比验证。所谓分析验证是指使用算法应用场景中的评价体系，对算法的处理结果进行评价与分析，并且衡量其结果是否符合系统设计的目标需求，图8给出了DVB-T系统在AWGN噪声信道下的接收信号频谱和星座图，在该系统中，两者分别使用信噪比（SNR）和调制误差率（MER）进行衡量。

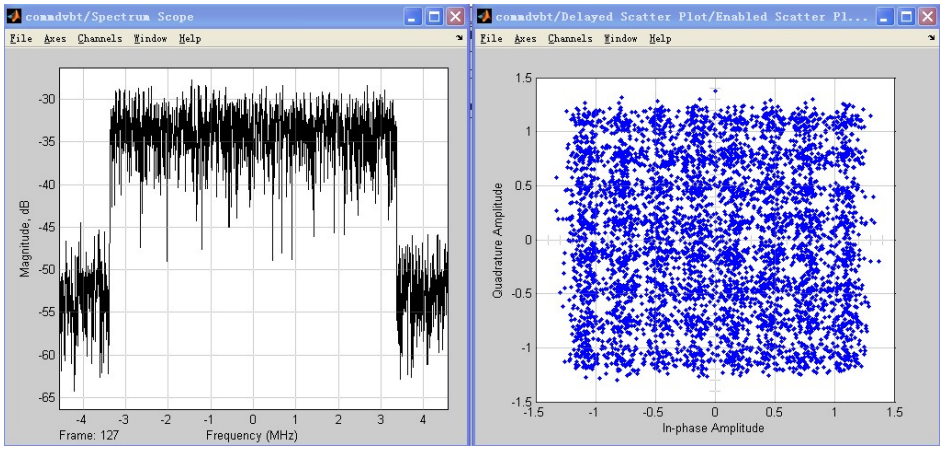

图 8: DVB-T系统仿真的频谱与星座图

所谓对比验证是指不直接分析与评价当前仿真模型的结果数据，而是将当前仿真的结果数据同以同样数据进行激励、抽象级别更高的可靠参考模型的结果数据进行对比，间接评价当前仿真模型输出结果的正确性及指标参数。

在系统的设计实现过程中，通常在抽象层次较高的层面使用分析验证的手段，比如同步数据流或是向量计算层面，因为这些层面的数据分析手段较为完备与便利，而在样点级层面，例如C程序或硬件描述语言（HDL）层面，通常采用对比验证的手段，即将相同激励信号下的算法的运行结果与高层次仿真的结果相比较并且通过评价计算误差来评估算法实现是否满足需求。

1.4.2 仿真数据传递策略

一般而言，低层次的仿真工具对数据进行抽象指标的分析和评价能力有限，并且底层模型的激励数据需要和高层次抽象模型的激励数据保持一致，所以在不同层次的仿真环境之间传递数据成为普遍需求，为解决此问题，通常使用文件或内存交互数据。

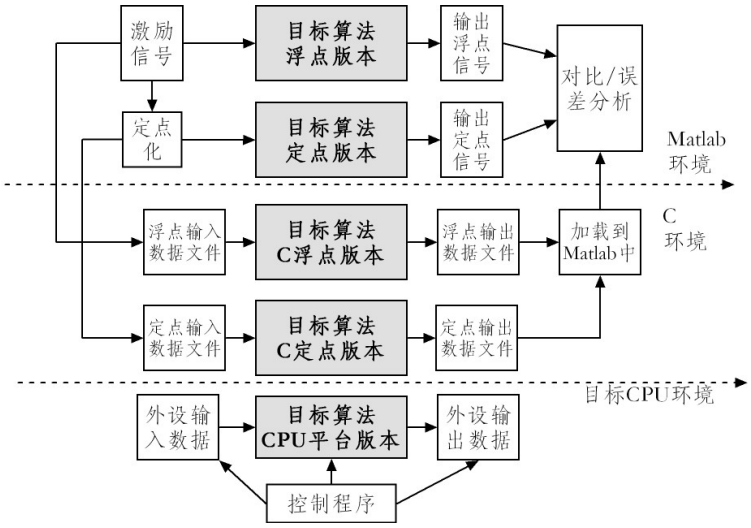

图 9: 文件数据传递

以开发嵌入式环境下的C算法程序为例，最终目标为运行于嵌入式处理器上的定点C程序，该算法从Matlab环境开始，经历浮点模型，定点模型，C环境下的浮点模型、定点模型，直至最终在嵌入式环境下进行运行调试。各种模型、包括嵌入式调试工具，均可以采用加载和导出数据文件的方法交互激励数据与结果数据，图9给出了基于文件传递数据的仿真结构。

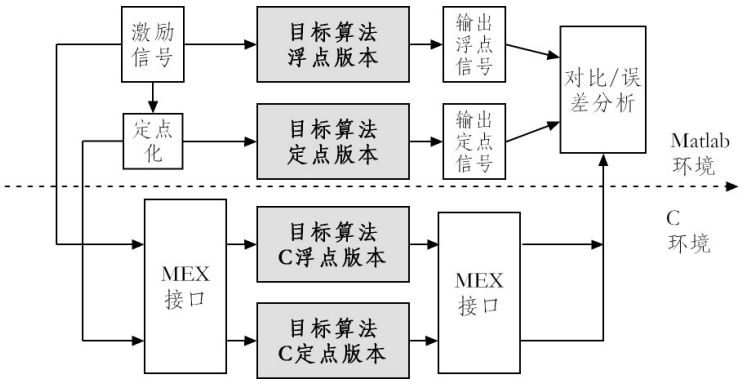

图 10: 内存数据传递

对于数据吞吐量较大的算法仿真，使用文件传递数据因其需要访问磁盘因而数据传递较慢，为提高传输效率，可以使用内存数据交互的方法，以Matlab为例，其提供了编程语言扩展接口（MEX）可以调用执行C、Fortran等编程语言的生成的计算模型代码，此种方法能够进行高速数据交互，但相比文件的方法，

其调试难度有所增加，图10给出了内存接口传递数据的仿真结构。

2 后记

作者水平有限，编此拙文，仅为抛砖引玉，希望这些粗浅的文字不至于误导读者，更希望能对您步入数字信号处理的有趣世界提供些许帮助，同时，作为一名在高校的研究院所工作的教师，作者也衷心的希望喜爱电子技术和数字通信、信号处理的本科同学报考我们数字工程中心的研究生。

这里是：中国传媒大学广播电视数字化教育部工程研究中心，在校内简称数字化工程中心，英文缩写是ECDAV，实验室的网址是http://ecdav.cuc.edu.cn 。我们是一个教育部直属的工程研究中心，请参考

http://baike.baidu.com/view/2391206.htm 中的第53号单位。

长久以来，我们致力于数字无线广播领域的研究，参与了相关领域多项国家标准的方案设计、样机研制、外场实验以及标准的起草，同时，我们也致力于专用领域的远程无线数据通信研究，这是因为数字广播中所采用的技术非常适合应用在数百公里的大尺度距离上的无中继数据传输，比如在突发的地质灾害现场，能够进行远程通联的设备往往只有短波电台和卫星，其它需要中继基站和路由的设备很多都不能发挥作用。

正因如此，在个人移动通信和无线互联网如此繁荣和吸引眼球的今天，数字广播仍然在无线通信领域拥有属于它的一席之地。同时，我们知道，无论是南海的堡礁或是北疆的哨所，亦或是深山村落里的油灯下，还是灾区简易房的褴褛里，祖国，需要我们，而我们，需要你。