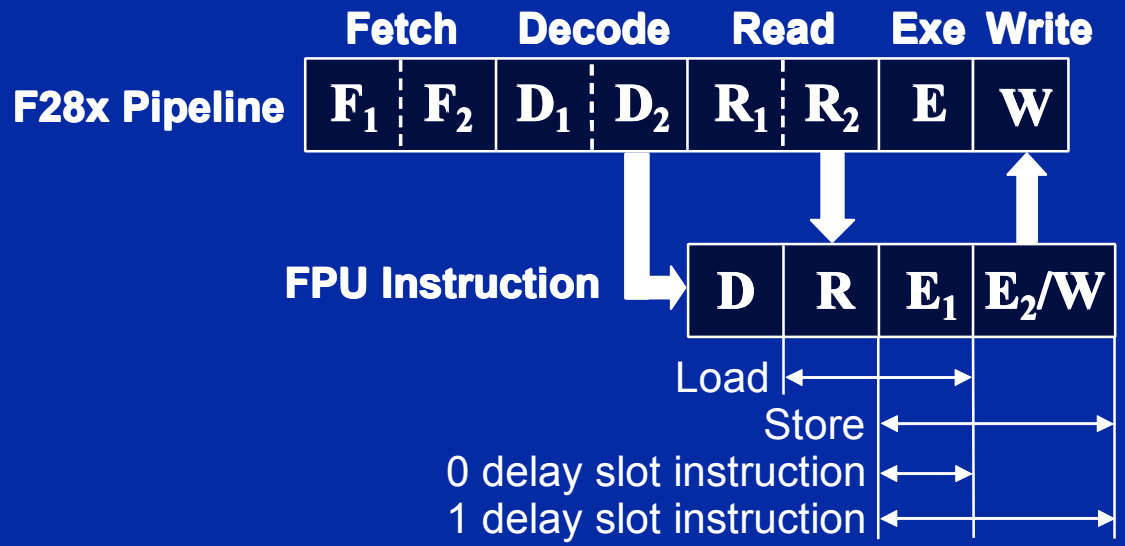
How to use CLA & FPU & VCU on 2806x

F28x CPU + FPU 流水线

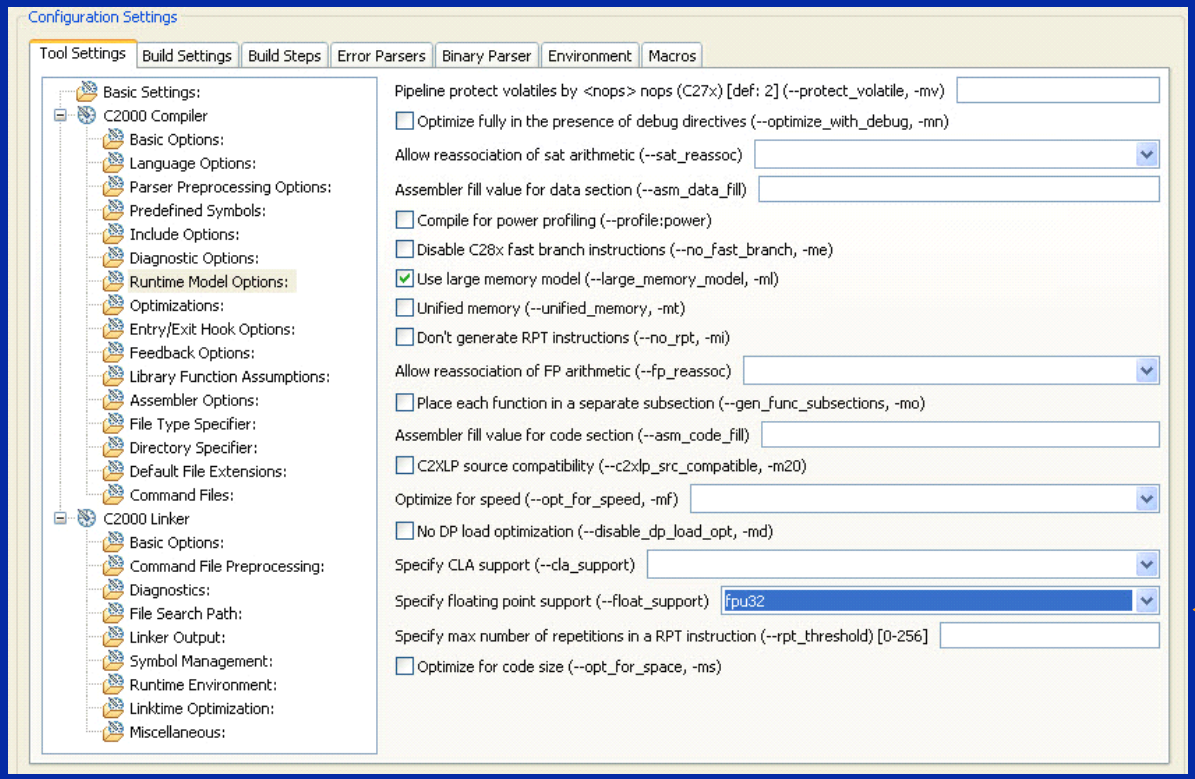


浮点数学运算及整数和浮点数之间的转换需要一级延迟 – 其他所有指令及运算都不需要延迟 **(load, store, max, min, absolute, negative, etc.)**

- 浮点运算单元有无保护流水线
  - 例如: **FPU** 可以在前一个指令写完结果之前发出新的指令
- 编译器避免流水线冲突
- 汇编程序检测流水线冲突
- 在浮点流水线延迟时段中执行非冲突指令可提高性能

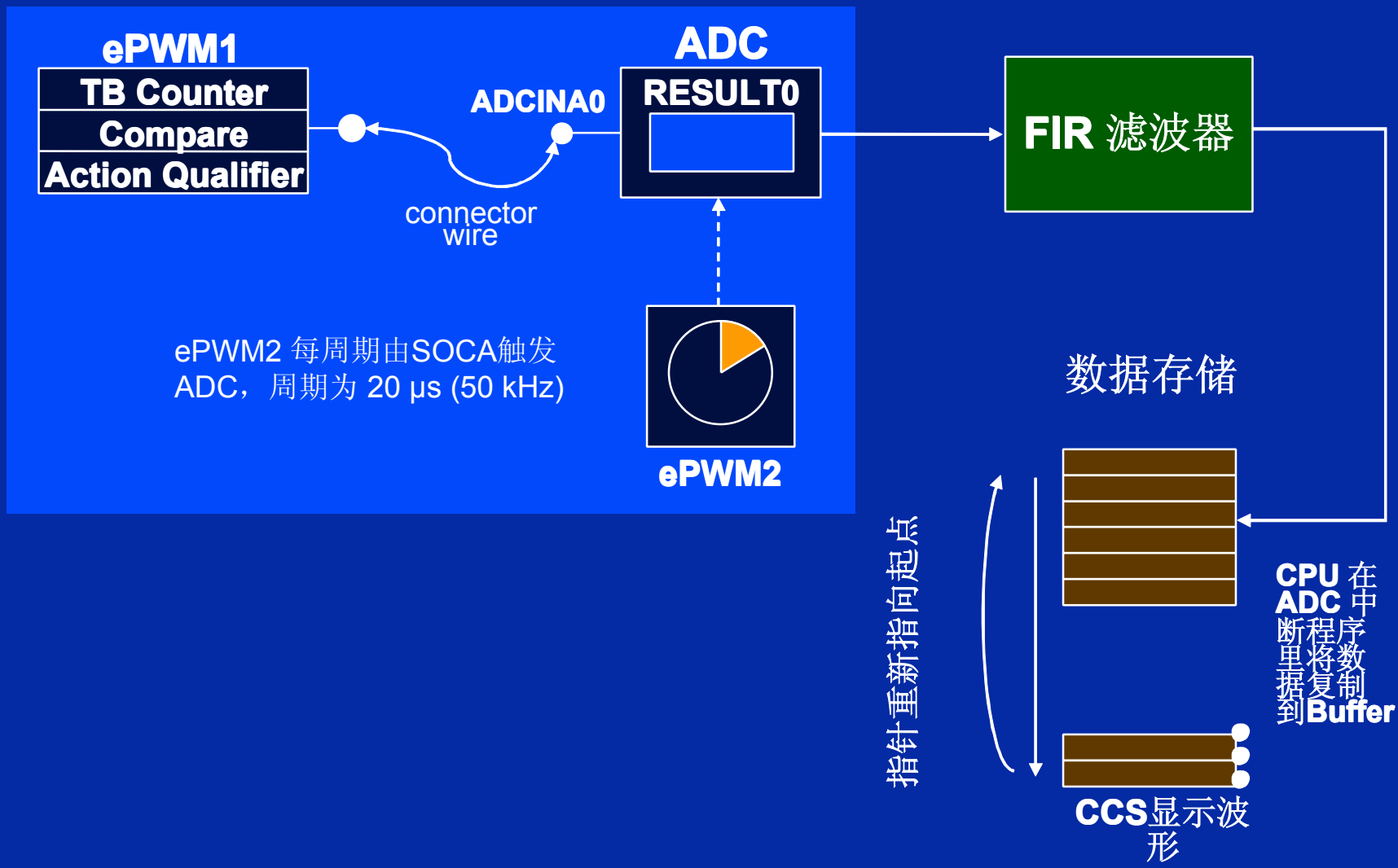
使用浮点运算

- 注意: 必须使用支持浮点的器件，如**2806x!**
- 新建**CCS**项目，选择浮点器件时，会自动添加浮点运算库 **FPU RTS library**，并且在编译参数设定中选择 **'fpu32'**以支持浮点

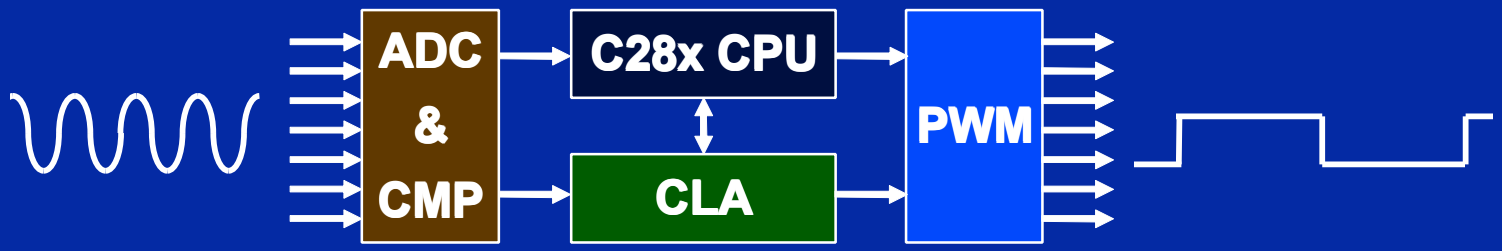


- 添加浮点库到 **CCS** 项目中
  - 标准库 **RTS lib** *(必须)*
    - **rts2800_fpu32.lib**
    - 和编译器一起添加
  - 高速 **RTS lib** *(可选)*
    - **C28x_FPU_FastRTS.lib**
    - **TI** 网站, **#SPRC664**
    - 更好的性能
    - *强烈推荐*
- 在**CCS**编译参数设定中，选择 **'fpu32'**

例程: IQmath & 浮点 FIR 滤波器



控制率加速器 (CLA)



- **CLA** 是一个独立的**32**位浮点数学运算加速器
- 独立执行算法， 与主**CPU**并行运算
- 直接访问 **ePWM / HRPWM, ADC** 结果和比较寄存器
- 可独立于**CPU**，回应外设中断
- 让**CPU**进行其他任务**(**通讯及分析判断**)**

CLA 初始化

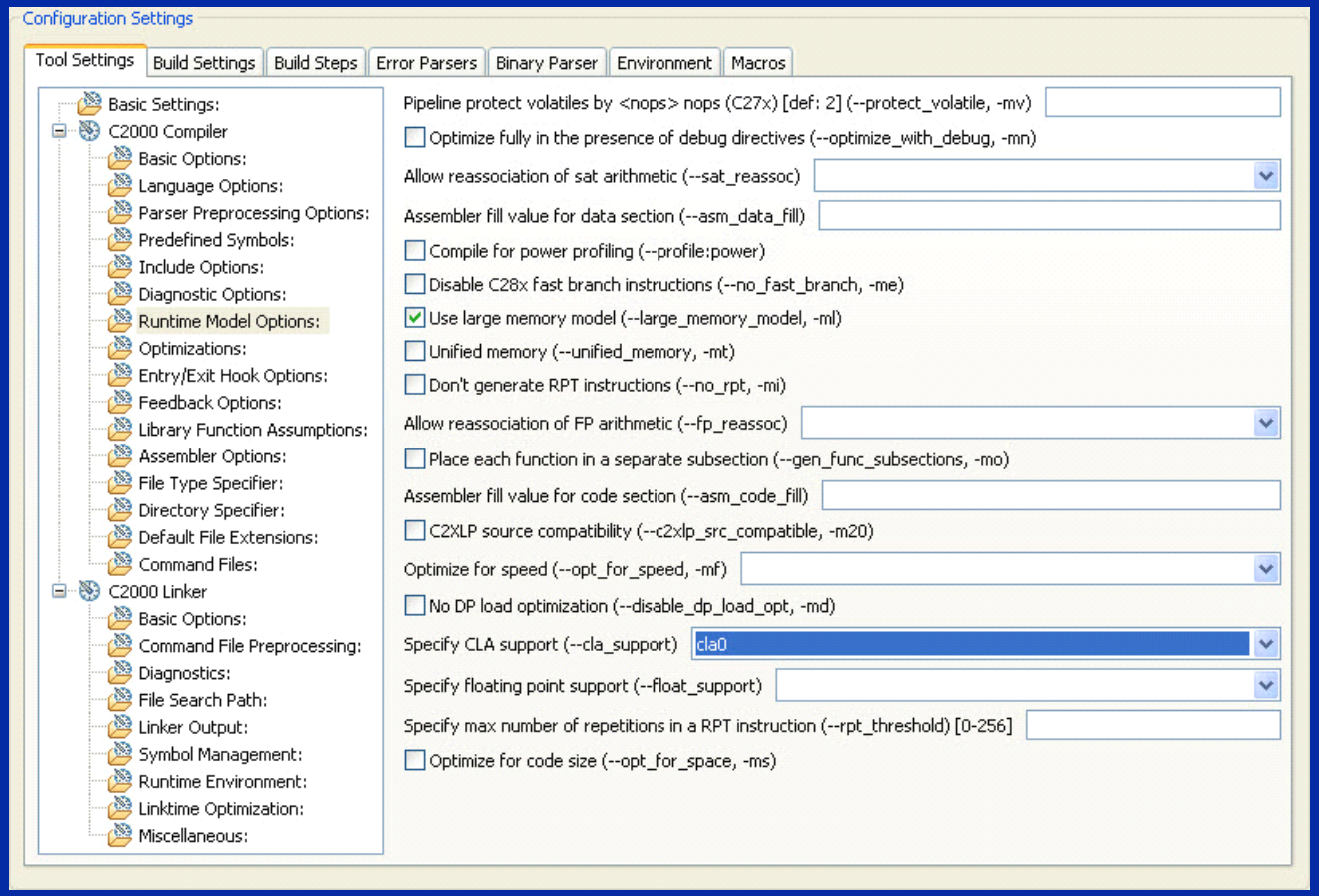
CLA 初始化由 CPU用 C 语言执行
(基本上由外设寄存器头文件完成)

1. 将 **CLA** 任务代码从 **flash** 复制到 **CLA** 程序 **RAM**
2. 根据需要初始化 **CLA** 数据 **RAMs**
   - 数据系数, 常量, 等等.
3. 配置 **CLA** 寄存器参数
   - 使能 CLA 时钟 (PCLKCR3 寄存器)
   - 配置 CLA 任务中断向量 (MVECT1-8 registers)
   - 选择需要的中断源 (PERINT1SEL register)
   - 如果需要, 使能 IACK 用软件启动任务
   - 映射 CLA 程序 RAM 及 data RAMs 到 CLA 空间
4. 在 **PIE** 中配置所需的 **CLA** 任务完成中断
5. 在 **MIER** 寄存器中使能 **CLA** 任务触发
6. 初始化 **ePWM** 及/或 **ADC** 触发 **CLA** 任务

数据在 CLA 和 CPU 间通过 message RAMs 传递

在 CCS 中使能 CLA 支持

- 注意: 必须使用包含 CLA 的 C28x 器件
- 创建 CCS 项目时，选择 CLA 器件会自动配置 “Specify CLA support” 选项: *‘cla0’*
- 这个设定是为了支持 CLA 代码编译



VCU 综述
C28x 扩展指令集支持

- **Viterbi** 运算
  - 用于通讯解码
- 复数计算
  - **16** 位定点复数 **FFT**
    - 用于扩展频谱通信, 以及许多信号处理算法
  - 复数滤波器
    - 用于改善数据可靠性, 传输距离及效率
  - 电力线载波通信 **(PLC)** 及雷达应用
- 循环冗余检验 **(CRC)**
  - 通讯及存储可靠性检验

VCU指令

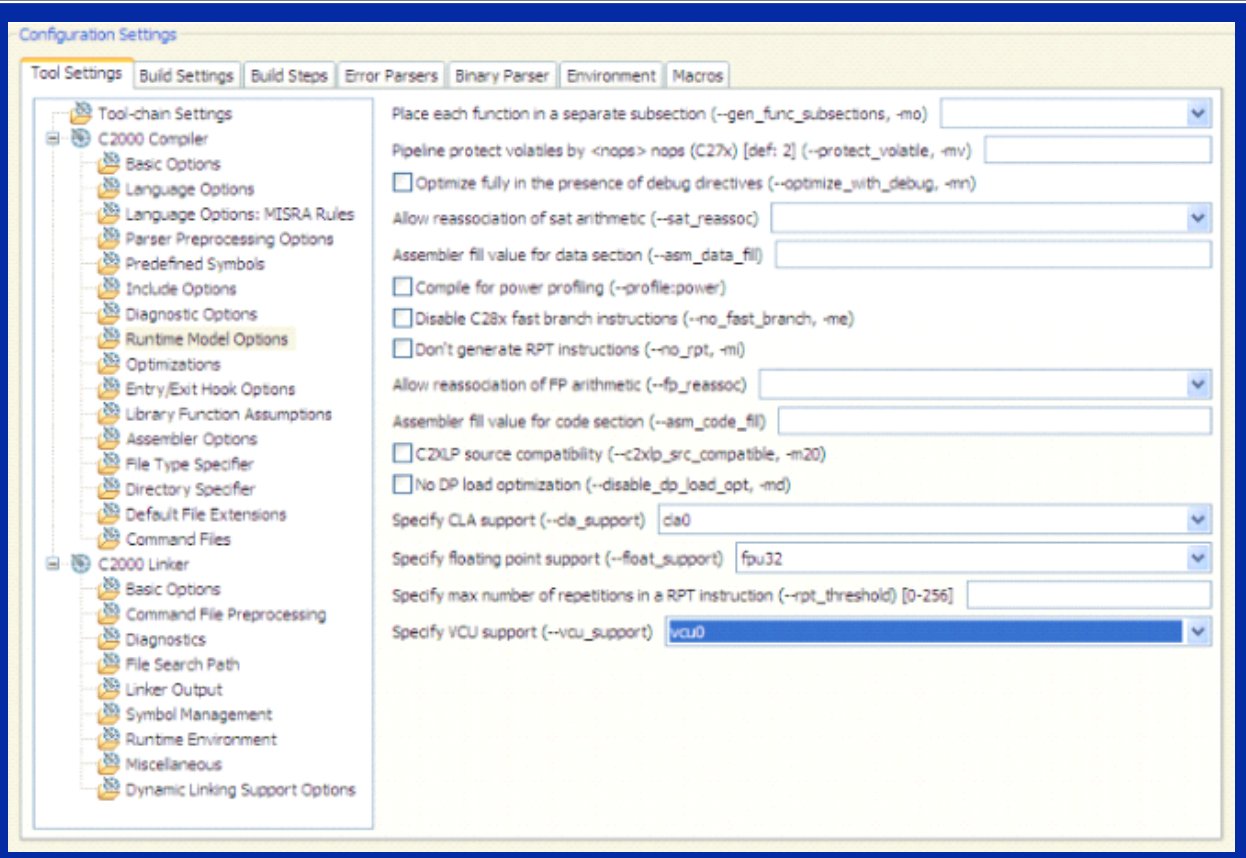
- 与 C28x 和 C28x+FPU相似的指令格式
  - 目标操作数总是在左边
  - 与 C28x 及 FPU 相同的指令，前面多一个“V”

```
CPU:    MPY      ACC,  T,    loc16
FPU:    MPYF32   R0H,  R1H,  R2H
VCU:    VCMPY    VR3,  VR2,  VR1,  VR0
```

目标操作数　源操作数

在CCS中使能VCU支持

- 注意：必须使用包含**VCU**的器件
- 新建**CCS**项目时，选择**VCU**器件时会自动设定**“Specify VCU Support”**选项：**“vcu0”**
- 必须设定此选项以编译**VCU**代码



FFT实现总结

- 16位FFT表现
  - C28x + VCU是最好的选择。幅值及相位计算与定点器件相同，因为VCU没有改进的算法。
- 32位定点FFT表现
  - 使用浮点器件，FPU可以让性能成倍增加。此外，幅值及相位计算能力也将显著提高。
  - 如果可以接受16位实现，可以考虑使用C28x + VCU，这将比一般定点器件更快。但是VCU没有关于幅值及相位的改进算法。这些运算操作在浮点器件上效率最高。
- 32位FPU vs 16位VCU
  - 两者表现差别不大， 但VCU没有关于幅值及相位的改进算法。
- CLA
  - CLA本身并不适合全部的FFT算法，但可考虑用来计算幅值及相位，这可以减轻主CPU的负担。例如：在2806x中，浮点FFT可以有C28x + FPU来完成，幅值及相位计算可以由CLA来完成。