好的，我们来详细规划一下，如何为40kHz信号在STM32F407VET6上设计并实现一个希尔伯特FIR滤波器算法，以得到一个正交（相移90°）的信号。

这个过程分为两大步：

1. **离线设计 (在PC上)**：计算出FIR滤波器的系数。
2. **在线实现 (在STM32上)**：用计算出的系数，通过高效的算法在MCU上实时运行滤波器。

第1步：离线设计滤波器系数 (PC端)

MCU只负责执行运算，滤波器的“灵魂”——系数，需要我们先在PC上用专业工具设计好。这里我们使用最流行和免费的Python + SciPy库来设计。

1.1 确定系统参数

- **信号频率 (f_signal)**: 40 kHz。这是我们关心的中心频率。
- **采样率 (Fs)**: 必须满足奈奎斯特定理 (`Fs > 2 * f_signal`)。为了让滤波器有良好的性能，我们通常取4-5倍以上。这里我们选择 **Fs = 200 kHz**。STM32F407的ADC完全可以轻松达到这个速率。
- **滤波器阶数 (N)**: 阶数越高，滤波器性能越好，但计算量越大。对于希尔伯特变换器，阶数**必须是奇数**。我们先从一个中等阶数开始，比如 **N = 31**。这个阶数对于STM32F407来说是小菜一碟。
- **带宽 (Bandwidth)**: 我们希望在哪个频率范围内实现90°相移。假设我们关心的是35kHz到45kHz的范围。

1.2 使用Python (SciPy) 生成系数

下面的Python脚本将根据以上参数，使用 `remez` 算法（也称Parks-McClellan算法）设计一个希尔伯特滤波器，并生成可以直接复制到C语言代码中的系数数组。

```python
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt

# --- 1. 定义滤波器参数 ---
N = 31                    # 滤波器阶数 (必须是奇数)
Fs = 200000               # 采样率 (Hz)
f_signal = 40000          # 信号频率 (Hz)
# 我们关心频带，例如从1kHz到采样率的一半以下
# 这里设置一个较宽的通带，例如从 5kHz 到 95kHz
band = [5000, 95000]

# --- 2. 使用remez算法设计希尔伯特滤波器 ---
# 对于希尔伯特变换器，我们定义一个通带，类型设置为'hilbert'
coeffs = signal.remez(N, band, [1], type='hilbert', fs=Fs)

# --- 3. 打印C语言格式的系数数组 ---
print("/* Hilbert FIR Filter Coefficients */")
print(f"/* N = {N}, Fs = {Fs} Hz */")
print(f"const float32_t hilbert_fir_coeffs[{N}] = {{")
for c in coeffs:
    print(f"    {c:.9f}f,")
print("};")

# --- 4. (可选) 绘制滤波器的频率和相位响应以供验证 ---
w, h = signal.freqz(coeffs, [1], worN=8000, fs=Fs)

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(8, 6))

# 幅度响应
ax1.plot(w / 1000, 20 * np.log10(np.abs(h)))
ax1.set_title('Filter Magnitude Response')
ax1.set_xlabel('Frequency (kHz)')
ax1.set_ylabel('Magnitude (dB)')
ax1.grid()
ax1.axvspan(band[0]/1000, band[1]/1000, color='lightgreen', alpha=0.5, label='Passband')
ax1.legend()

# 相位响应
angles = np.unwrap(np.angle(h)) * 180 / np.pi
ax2.plot(w / 1000, angles)
ax2.set_title('Filter Phase Response')
ax2.set_xlabel('Frequency (kHz)')
```

```
ax2.set_ylabel('Phase (degrees)')
ax2.grid()
ax2.axvspan(band[0]/1000, band[1]/1000, color='lightgreen', alpha=0.5)
ax2.set_yticks([-180, -90, 0, 90, 180])

fig.tight_layout()
plt.show()
```

运行这个脚本，它会打印出C代码格式的系数，并且显示出滤波器的响应曲线。你应该能看到在通带内，幅度响应接近0dB，相位响应接近-90°。

第2步：在线实现滤波器 (STM32F407端)

我们强烈推荐使用ARM官方为Cortex-M内核提供的 **CMSIS-DSP库**，它针对F407的DSP指令和FPU进行了高度优化，性能远超自己写的普通C语言循环。

2.1 项目配置

1. 在你的STM32CubeIDE或Keil项目中，包含CMSIS-DSP库。在CubeMX中，可以在 "Software Packs" -> "ARM::CMSIS" 中选择包含 "DSP" 组件。
2. 在你的C代码文件中，包含头文件：`#include "arm_math.h"`。

2.2 编写滤波代码

```
#include "arm_math.h"

// --- 1. 从Python脚本中复制系数 ---
/* Hilbert FIR Filter Coefficients */
/* N = 31, Fs = 200000 Hz */
const float32_t hilbert_fir_coeffs[31] = {
    0.057390235f,
    0.000000000f,
    -0.071742423f,
    0.000000000f,
    0.091763787f,
    0.000000000f,
    -0.123539343f,
    0.000000000f,
    0.181822509f,
    0.000000000f,
    -0.297405213f,
    0.000000000f,
    0.628419638f,
    0.000000000f,
    -1.574676633f,
    0.000000000f,
    1.574676633f,
    0.000000000f,
    -0.628419638f,
    0.000000000f,
    0.297405213f,
    0.000000000f,
    -0.181822509f,
    0.000000000f,
    0.123539343f,
    0.000000000f,
    -0.091763787f,
    0.000000000f,
    0.071742423f,
    0.000000000f,
    -0.057390235f,
};

// --- 2. 定义滤波器所需的变量 ---
#define FIR_ORDER 31
```

```c
#define BLOCK_SIZE 1 // 每次处理一个采样点

static arm_fir_instance_f32 fir_instance;       // FIR滤波器实例结构体
static float32_t fir_state[FIR_ORDER + BLOCK_SIZE - 1]; // FIR状态缓冲区，用于存放历史采样

// --- 3. 初始化滤波器 ---
void hilbert_fir_init(void) {
    // 调用CMSIS-DSP库的初始化函数
    arm_fir_init_f32(&fir_instance, FIR_ORDER, (float32_t *)hilbert_fir_coeffs, fir_state, BLOCK
}

// --- 4. 运行滤波器 ---
// 这个函数应该在每次ADC采样完成后被调用
float32_t hilbert_fir_process(float32_t input_sample) {
    float32_t output_sample;

    // 调用CMSIS-DSP库的处理函数
    // 输入一个点，输出一个点
    arm_fir_f32(&fir_instance, &input_sample, &output_sample, BLOCK_SIZE);

    return output_sample;
}

/*
// --- 5. 在主程序或中断中调用 ---
// 伪代码示例
void ADC_ConversionComplete_Callback() {
    // 从ADC获取原始采样值，并进行必要的转换（例如从0-4095转为-1.0到1.0的浮点数）
    float32_t original_signal = get_adc_sample_as_float();

    // 通过FIR滤波器得到正交信号
    float32_t quadrature_signal = hilbert_fir_process(original_signal);

    // 在这里使用 quadrature_signal 进行后续处理...
}

int main(void) {
    // ... 系统初始化 ...
    hilbert_fir_init(); // 初始化滤波器
    start_adc_with_dma(); // 启动ADC连续采样

    while(1) {
        // ... 主循环任务 ...
```

```
    }
}
*/
```

总结与验证

1. **设计**：在PC上使用Python等工具，根据你的采样率和性能要求（阶数、带宽）设计滤波器，生成系数。
2. **实现**：在STM32F407上，利用其强大的Cortex-M4内核和FPU，通过CMSIS-DSP库来实现滤波算法。将PC上生成的系数数组复制到代码中，调用库提供的 `init` 和 `process` 函数即可。
3. **验证**：要验证效果，你可以使用STM32的DAC功能。
   - **DAC通道1** 输出原始的40kHz信号（可以由定时器+内部sin函数生成一个测试信号）。
   - **DAC通道2** 输出经过希尔伯特FIR滤波器处理后的信号。
   - 用双通道示波器同时观察这两个DAC通道的输出，你应该能看到两个频率相同、幅度接近的波形，且它们之间有**精确的90度相位差**。