1.FPGA与单片机的区别？

单片机和 FPGA的区别，本质是软件和硬件的区别。
单片机设计属软件范畴；它的硬件（单片机芯片）是固定的，通过软件编程语言描述软件指令在硬件芯片上的执行；
FPGA设计属硬件范畴，它的硬件（ FPGA）是可编程的，是一个通过硬件描述语言在 FPGA芯片上自定义集成电路的过程；
二者最大的区别：单片机（无论哈佛总线结构或者冯诺依曼结构）均为取出指令 -> 执行，指令是顺序执行的（即使是中断，其发生后的中断服务程序也是顺序执行的）；而 FPGA（包括 CPLD）是基于逻辑门和触发器的，它是并行执行方式，即当CLOCK上升沿到来时，所有的触发器都会动作，它没有取出指令 ->执行这种操作，数字电路中所有逻辑门和触发器（ D,SR等）均可以实现，它适合真正意义上的并行任务处理。

2.单片机、 FPGA、 DSP、 ASIC的区别

1、 ASIC原本就是专门为某一项功能开发的专用集成芯片，集成度很低，成本很低，可是够用了。后来 ASIC发展了一些，称为半定制专用集成电路，相对来说更接近 FPGA，甚至在某些地方， ASIC就是个大概念， FPGA属于 ASIC之下的一部分。

2、 FPGA基本就是高端的 CPLD，数字电路。这种器件是用逻辑门来表述性能的。本身他就是一堆的逻辑门，与非门、或非门、触发器（可以用与非门形成吧）等基本数字器件，编程决定了有多少器件被使用以及它们之间的连接。通过硬件描述语言把它转成电路连接，从最基本的逻辑门层面上连接成电路（参见数字电路书上那些全加器触发器什么的）。应该说，虽然看起来像一块 CPU，其实是完全硬件实现的。它是在 PAL、GAL、EPLD等可编程器件的基础上进一步发展的产物。它是作为专用集成电路 (ASIC)领域中的一种半定制电路而出现的，既解决了定制电路的不足，又克服了原有可编程器件门电路数有限的缺点。 FPGA在抗干扰，速度上有很大优势。

后来因为写代码麻烦，对控制部分比较薄弱，本来跟其他 CPU配合使用，即麻烦的算法 CPU提交给 FPGA，FPGA算完把结果再返回给 CPU。可是这样外围电路就变得麻烦。

于是提出了 SOC设计方法，就是直接在 FPGA里写一个 CPU出来，既然 FPGA万能，做个 CPU自然毫无压力。

这其中还有软核和硬核的区别，不过除了性能，使用方法大同小异。所谓 IP 核，就是把各种专用集成电路用硬件描述语言描述，然后烧到 FPGA里形成专门的电路，这样就不必另外搭芯片了，所有的电路在一片 FPGA里面形成。

3、 DSP实际应该称为 DSPs，即用于 DSP处理的专用芯片。跟普通计算机的区别一方面是他是哈佛结构的，也就是数据和程序空间分开（普通计算机是冯诺依曼结构）。另一方面他有流水线结构，不过现在其他也有了，见贤思齐。再一方面他有专用的硬件算法电路，用以完成 DSP运算，比如最基本的乘法累加。上过 DSP的就知道，蝶形算法 FFT什么的，拆成最基本单元就是乘法累加，把这部分加速了，整体性能就有非常大的提高。 DSP对于

流媒体的处理能力远远的优于通用 CPU。所以你看现在手机 CPU，至少语音部分都是用 DSP的。后来 DSP概念也复杂化，各家都把一个控制核心整合到 DSP里面，比如现在的智能手机芯片。可以看一下高通或者 TI 的片，基本是一个 ARM 核控制整体运算，一个 DSP 处理语音编解码，一个 GPU负责图像运算，一个基带和天线处理模块负责通信，再加一些七七八八的东东比如 GPS模块什么的。

4、单片机就是一个百搭的通用 CPU，是集成在单一芯片上的微型计算机系统，麻雀虽小可是五脏俱全，也有运算器、控制器、存储器、总线及输入输出设备，采用也是存储程序执行的方式，对单片机的编程就是对其中的 ROM 写入程序，在加电后 ROM 中的程序会像计算机内存中的程序一样得到逐条的执行。单片机计算速度和性能有限，但在一些基本控制上搓绰绰有余。

单片机提供各种接口来对整体进行控制，相当一个总调度，当然，简单的功能一片 CPU独立工作也就完成了。原来的 51 系列就是一堆 IO 口，后来慢慢的把常用的 PWM,AD 之类的功能加入了单片机。主要包括用了无数年仍然牛逼各大学必教的 51 系列，还有 AVR,PIC,ARM, HOTEK……其实 ARM9 以后，已经说不清 ARM 算哪类了，目前的架构来看，更接近 DSP。

单片机和嵌入式通过主函数或者操作系统来实现任务调度的途径来响应各种外部条件触发，并通过软件输出相应的状态来实现电子系统的正常工作。 FPGA是通过纯硬件来实现各种激励的相应的。

3.FPGA与 DSP的比较

DSP和 FPGA是嵌入式开发处理器的三大巨头之二，很多刚刚接触嵌入式的朋友都会心存疑问，到底 DSP和 FPGA哪个牛一点，学哪种好一点？ FPGA与 DSP相比较，哪个更有前途？今天，我就以自己的经验，和大家通俗介绍一下吧：

FPGA是英文 Field Programmable Gate Array（现场可编程门阵列）的缩写，它是在 PAL、GAL、PLD等可编程器件的基础上进一步发展的产物，是专用集成电路（ ASIC）中集成度最高的一种。 FPGA采用了逻辑单元阵列 LCA（Logic Cell Array）这样一个新概念，内部包括可配置逻辑模块 CLB（Configurable Logic Block）、输出输入模块 IOB（Input Output Block）和内部连线（ Interconnect ）三个部分。用户可对 FPGA内部的逻辑模块和 I/O 模块重新配置，以实现用户的逻辑。它还具有静态可重复编程和动态在系统重构的特性，使得硬件的功能可以像软件一样通过编程来修改。作为专用集成电路（ ASIC）领域中的一种半定制电路， FPGA既解决了定制电路的不足，又克服了原有可编程器件门电路数有限的缺点。可以毫不夸张的讲， FPGA能完成任何数字器件的功能，上至高性能 CPU,下至简单的 74 电路，都可以用 FPGA来实现。 FPGA如同一张白纸或是一堆积木，工程师可以通过传统的原理图输入法，或是硬件描述语言自由的设计一个数字系统。通过软件仿真，我们可以事先验证设计的正确性。在 PCB完成以后，还可以利用 FPGA的在线修改能力，随时修改设计而不必改动硬件电路。使用 FPGA来开发数字电路，可以大大缩短设计时间，减少 PCB面积，提高系统的可靠性。

DSP（digital singnal Processor）是一种独特的微处理器，有自己的完整指令系统，是以数字信号来处理大量信息的器件。一个数字信号处理器在一块不大的芯片内包括有控制单元、运算单元、各种寄存器以及一定数量的存储单元等等，在其外围还可以连接若干存储器，并可以与一定数量的外部设备互相通信，有软、硬件的全面功能，本身就是一个微型计算机。 DSP采用的是哈佛设计，即数据总线和地址总线分开，使程序和数据分别存储在两个分开的空间，允许取指令和执行指令完全重叠。也就是说在执行上一条指令的同时就可取出下一条指令，并进行译码，这大大的提高了微处理器的速度。另外还允许在程序空间和数据空间之间进行传输，因为增加了器件的灵活性。其工作原理是接收模拟信号，转换为 0 或 1 的数字信号，再对数字信号进行修改、删除、强化，并在其他系统芯片中把数字数据解译回模拟数据或实际环境格式。它不仅具有可编程性，而且其实时运行速度可达每秒数以千万条复杂指令程序，远远超过通用微处理器，是数字化电子世界中日益重要的电脑芯片。它的强大数据处理能力和高运行速度，是最值得称道的两大特色。由于它运算能力很强，速度很快，体积很小，而且采用软件编程具有高度的灵活性，因此为从事各种复杂的应用提供了一条有效途径。根据数字信号处理的要求，

上面都是一些基本概念的介绍，下面我就来通俗介绍一下，在 DSP里，你是一个软件设计者，硬件已经完全固化，你所要做的，就是在这个固定的硬件平台实现其功能的最优化，一般 TI 的 DSP涉及最多的是一些基本的 BIOS操作系统之间的任务调度，以及算法改进与优化等待， DSP的关键优势包括其对于新型及复杂算法时的更短的开发时间，以及能够运行多种算法的灵活性。

而对于 FPGA来说，你是一个硬件设计者， FPGA就是一张白纸，上面写什么，画什么都取决于你。同样一片 FPGA，菜鸟和高手实现的功能会是天壤之别， FPGA的最大优势在于硬件实现已及通过并行处理实现的效率增益。使用 FPGA，您大多的时间并非进行算法设计与优化，而是逻辑设计与时序约束等等。

下面再举一个最通俗的例子，同样使用 FPGA与 DSP，对图像进行处理，这里的算法采用中值滤波，中值滤波是数字图像处理中十分常见也是非常有用的一种图像处理算法，其基本步骤如下：

中值滤波法将每一象素点的灰度值设置为该点某邻域窗口内的所有象素点灰度值的中值 .

实现方法：

1：通过从图像中的某个采样窗口取出奇数个数据进行排序

2：用排序后的中值取代要处理的数据即可

中值的计算在于对滑动窗口内像素的排序操作。要进行排序，就必须对序列中的数据像素做比较和交换，数据元素之间的比较次数是影响排序速度的一个重要因素。传统的排序串行算法是基于冒泡排序法，若窗口内像素为 m 个，则每个窗口排序需要做 m(m-2)/2 次像

素的比较操作，时间复杂度为　O(M 平方)。此外，常规的滤波算法使窗口每移动一次，就要进行一次排序，这种做法实际上包含了大量重复比较的过程。若一幅图像的大小为　N*N ，则整个计算需要　O(M 平方)时间，当窗口较大时计算量很大，较费时。

以下是采用　TI C6000系列的 DM642 上，采用内联函数优化过的中值滤波算法

```
voidIMG_median_Row( const unsigned short *restrict i_data, int n, unsigned  short *restrict o_data)

{

constint     *line0,      *line1,      *line2;

int             *line_y;

inti;

unsignedint     R00,         R10,         R20;

unsignedint     R01,         R11,         R21;

unsignedint     x0_01,       x1_01,       x2_01;

unsignedint     R_max1,      R_min1,      R_min2;

unsignedint     row0_pk,     row1_pk,     row2_pk;

unsignedintmin_max,     med_med,     max_min;

unsignedint     med_max1,    med_min1,    med_min2;

unsignedint     median2;



line0  = (int *) i_data;

line1  = (int *) (i_data + n);

line2  = (int *) (i_data + n * 2);

line_y = (int *) o_data;


    R00    = 0x00000000;

    R10    = 0x00000000;

    R20    = 0x00000000;
```

```
_nassert(          n       >= 4);

_nassert(          n  % 4  == 0);

_nassert((int) line0  % 8  == 0);

_nassert((int) line1  % 8  == 0);

_nassert((int) line2  % 8  == 0);

_nassert((int) line_y % 8  == 0);


#pragma MUST_ITERATE(2,,2)

#pragma UNROLL(2)

for (i = 0 ; i< n; i += 2)

   {


      x0_01    = *line0++;

      x1_01    = *line1++;

      x2_01    = *line2++;



      R_max1   = _max2(x0_01,x1_01);

      R_min1   = _min2(x0_01,x1_01);

      R_min2   = _min2(R_max1,x2_01);


      R01      = _max2(R_max1,x2_01);

      R11      = _max2(R_min1,R_min2);

      R21      = _min2(R_min1,R_min2);


      row0_pk  = _packlh2(R01,R00);

      row1_pk  = _packlh2(R11,R10);

      row2_pk  = _packlh2(R21,R20);
```

```
min_max  = _min2(R00,R01);
min_max  = _min2(min_max,row0_pk);

        med_max1 = _max2(R10,R11);
        med_min1 = _min2(R10,R11);
        med_min2 = _min2(med_max1,row1_pk);
med_med  = _max2(med_min1,med_min2);

max_min  = _max2(R20,R21);
max_min  = _max2(max_min,row2_pk);

        med_max1 = _max2(min_max,med_med);
        med_min1 = _min2(min_max,med_med);
        med_min2 = _min2(med_max1,max_min);
median2  = _max2(med_min1,med_min2);

        *line_y++ = median2;

        R00 = R01;
        R10 = R11;
        R20 = R21;

    }
}

voidMedianFliter(const unsigned short *restrict Image_In, unsigned short *restrict
Image_Out,intRow,int Column)
{
inti;
```

```
for(i = 0; i< Row; i++)
    {
IMG_median_Row((Image_In+ i * Column), Column, (Image_Out+ i * Column));
     }
}
```

再看看 ALTERA FPGA实现的中值滤波，共用 LE 692个， FIFO，D 触发器若干，代码约 4000 行… ..

对比一下速度， 3*3 中值滤波器，图像大小 1024*768, 灰度图像

执行效果 DSP——帧延时 >1 帧，计算时间未优化前 C 代码 24 毫秒算法流程，循环等优化后 7 毫秒内联函数优化后 1.6 毫秒线性汇编优化后 272 微秒

执行效果 FPGA——帧延时 =2 行像素，计算时间由输入时钟定，如果像素时钟大于 50M 整个图像处理时间不足 1 微秒，但是像素时钟受整个系统时序的约束，过快会使逻辑工作在不稳定状态。

现在大家估计也知道 FPGA与 DSP的最大区别了吧，呵呵。 DSP——编程速度快，方便，适合做算法验证，如果想用好 DSP，那么大部分时间都在做算法与语言优化工作。 FPGA——编程速度慢，实现麻烦，不适合做算法验证，但是一旦实现后，可以进行流水线操作，延时非常低。

综上所述， FPGA与 DSP优缺点十分明显，所以现在音视频处理，移动通信或者整个通信行业等大量信号处理的工程项目中，流行的解决方案都是 FPGA+DSP，FPGA做逻辑控制,DSP做浮点算法，如果算法不是很占资源的，也有直接用 FPGA来做的，但两大 FPGA厂商都最近都推出了带 DSP平台的 FPGA产品，以后 FPGA与 DSP的界限将越来越模糊，会慢慢的合二为一，总之，目前而言，由 DSP和 FPGA结合而成的混合式方案常常能够为高性能多处理应用提供最好的方案，让每个器件都发挥其作用。 FPGA和 DSP是两项互补的技术，而不是互相竞争的对手。对于长远来看，我认为是个殊途同归的过程，最后的产物，到底叫 FPSP还是叫 DSGA，那就要看各位 IC厂商的造化了，呵呵……