**AD9361 和Zynq**

**及其参考设计说明**

作者： 刘恒良 时间： 2016年3月1日

目录

[一、AD9361概述 3](#_Toc444701201)

[1.1 AD9361芯片结构 3](#_Toc444701202)

[1.2 AD9361性能特点 3](#_Toc444701203)

[1.3 AD-FMCOMMS2-EBZ性能特点 3](#_Toc444701204)

[二、Zynq-7000概述 4](#_Toc444701205)

[2.1 Zynq的芯片结构 4](#_Toc444701206)

[2.2 ZC702简介和结构 6](#_Toc444701207)

[三、AD9361和ZC702之间的数据通路 7](#_Toc444701208)

[四、AD9361参考设计说明（PL侧硬件部分） 9](#_Toc444701209)

[4.1 IP核的概念 9](#_Toc444701210)

[4.2 硬件设计 9](#_Toc444701211)

[五、AD9361 参考设计说明（PS侧软件部分） 10](#_Toc444701212)

[5.1 AD9361 no-OS Software 概述 10](#_Toc444701213)

[5.2 AD9361 no-OS Software 顶层目录说明 10](#_Toc444701214)

[5.3 main.c文件 11](#_Toc444701215)

[5.4 dac\_init函数 13](#_Toc444701216)

[5.5 adc\_capture函数 16](#_Toc444701217)

# 一、AD9361概述

## 1.1 AD9361芯片结构

AD9361是一款面向3G和4G基站应用的高性能、高集成度的射频(RF)Agile Transceiver™捷变收发器。芯片采用了零中频架构，将整个射频以及中频信号电路集成在一个芯片中，包括射频放大器、 模拟滤波器、混频器、解调器、 12 位的 ADC 和 DAC 的 RF 2x2（接收和发送通道各两路） 收发器， 另外还集成了收发通道的频率合成器， 同时为每个接收子系统集成了独立的自动增益控制( AGC) 、直流失调校正、正交校正和数字滤波电路，消除了数字基带中提供这些功能的必要性，每个通道搭载两个高动态范围 ADC， 先将收到的 I 信号和 Q 信号进行数字化处理，然后将其传给可配置抽取滤波器和抽头有限脉冲响应( FIR) 滤波器，以相应的采样率生成 12 路输出信号。

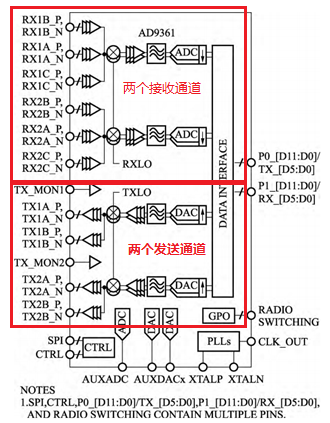


图 1 AD9361芯片结构

## 1.2 AD9361性能特点

AD9361 的工作频率为 70 MHz ～ 6 GHz，涵盖了大部分特许执照和免执照频段，支持可调谐 200 kHz ～ 56 MHz 的通道带宽，且具有高度的可编程能力， 发射器采用了直接变频架构，可实现较高的调制精度和较低的噪声，在接收通道，接收噪声系数可以做到小于 2. 5 dB，此外，该芯片的 EVM 可以做到小于-40 dB，可为外部功率放大器的选择留出客观的系统裕量，并且，芯片还支持 AGC 自动增益和更加灵活的手动增益模式，支持外部控制。

**1.3 AD-FMCOMMS2-EBZ性能特点**

AD-FMCOMMS2-EBZ是一款高速模拟模块，设计用于展示AD9361的硬件评估板。AD-FMCOMMS2-EBZ使射频工程师能够将AD9361连接到射频测试平台(矢量信号分析仪、信号发生器等)并测量性能。

图 2为AD-FMCOMMS2-EBZ的外观：

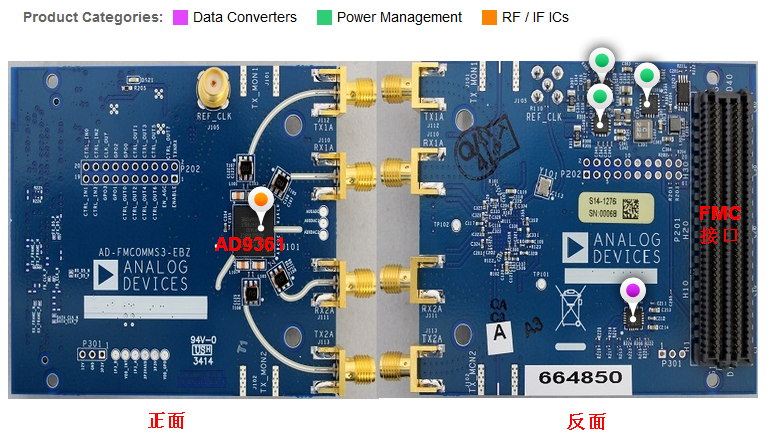


图 2 AD-FMCOMMS2-EBZ外观

通俗来讲，AD9361就是一个集成度很高的信号收发器，用户不需要向普通收发器那样准备放大器，混频器等等部件；它可以通过搭载它的评估板上的FMC接口与各种母板连接，连接后，在母板上用户可以使用ADI公司提供的API编程来配置AD9361的一些参数例如滤波器系数等等，达到灵活控制收发器的目的，而且工作频率宽，设计也很简单。

AD-FMCOMMS2-EBZ就是是搭载AD9361的评估板，用于外围设备（例如天线和FMC接口）与AD9361的连接。

**二、Zynq-7000概述**

**2.1 Zynq的芯片结构**

Zynq-7000 AP SOC是Zynq-7000全可编程片上系统的缩写（Zynq-7000 All Programmable System on Chip），它通过将一个双核ARM Cortex-A9处理系统（Processing System，简称PS）和Xilinx 7-Series 28nm 可编程逻辑（Programmable Logic，简称PL）及各种接口等周边设备集成到一个芯片上组成一个片上系统（SOC），来减少系统的复杂性。图 3为一个最简单的Zynq结构模型：

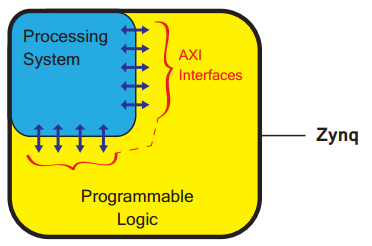


图 3 Zynq简单结构模型

Zynq最为简单的结构模型就只有两部分：PS和PL，PS和PL之间通过AXI接口进行数据通信，这样，使用Zynq SOC既可以单独使用ARM来实现嵌入式系统的设计，又可以使用FPGA来实现各种时序和逻辑的设计，最为关键的是可以同时使用二者来进行更为灵活的系统设计。

值得注意的是，Zynq的处理系统（PS）并不是只包含ARM处理器，还包含应用处理单元（Application Processing Unit，APU）和外围接口，缓存区，内存接口和时钟电路；Zynq可以单独使用PS部分，但无法单独使用PL部分，要想使用PL部分必须得启动PS侧，通过PS来配置PL。

图 4为较为详细的Zynq结构图，上半部分PS侧结构，其中为绿色部分为APU；下半部分为PL侧结构。

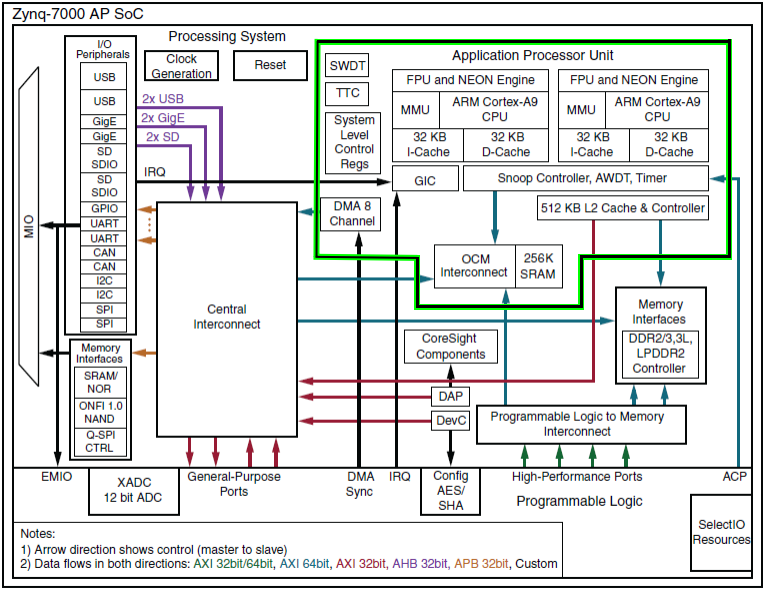


图 4 Zynq结构模型

**2.2 ZC702简介和结构**

Xilinx ZC702是一个评估板，为Zynq SoC提供一个开发和评估的硬件环境。ZC702和一些嵌入式处理系统有共同的特性，包括DDR3内存，三模式以太网，通用I/O和两个URAT接口。另外ZC702还支持FMC。使用ZC702作为平台连接AD-FMCOMMS2-EBZ板子可以生成需要发送的信号、对采集到的信号进行处理等功能。图 5为ZC702的外观结构，图 6为ZC702的简略的框架结构

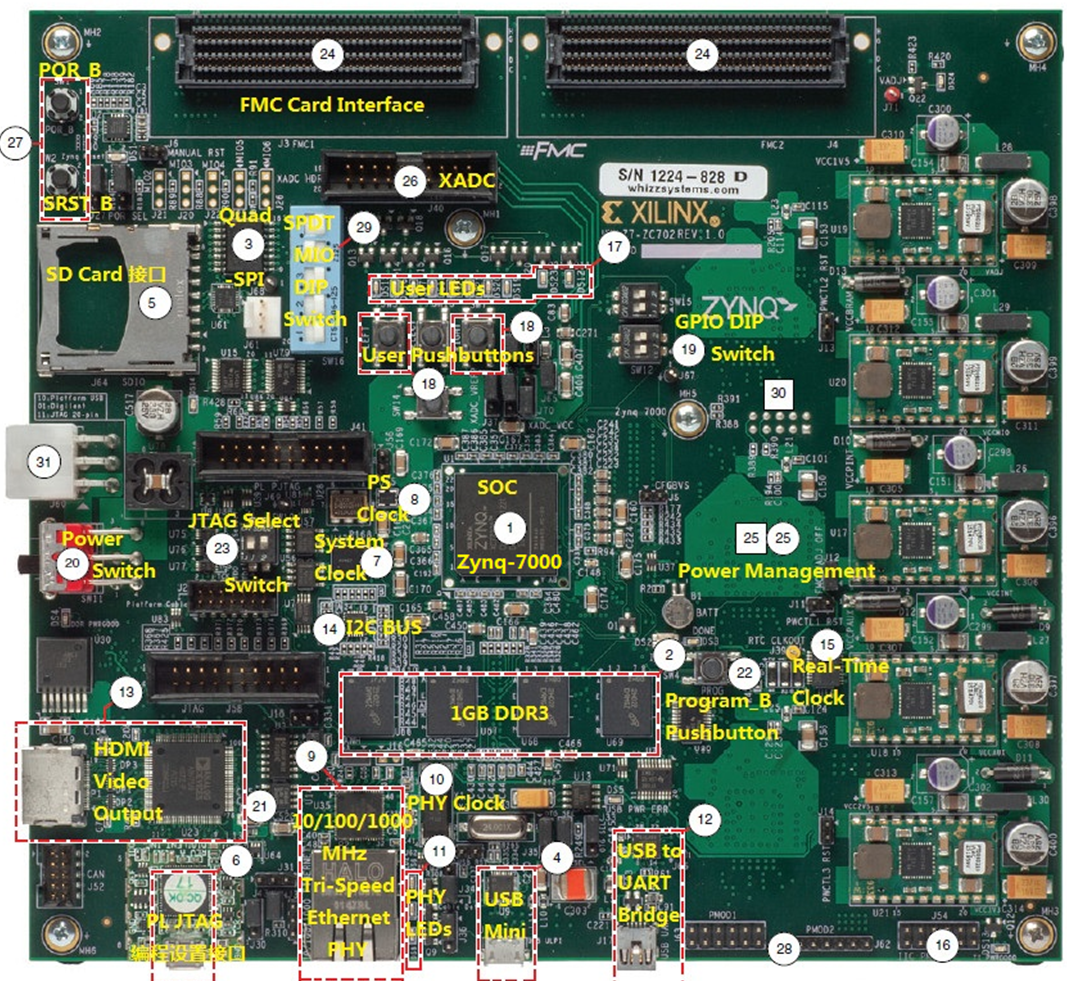


图 5 ZC702外观结构

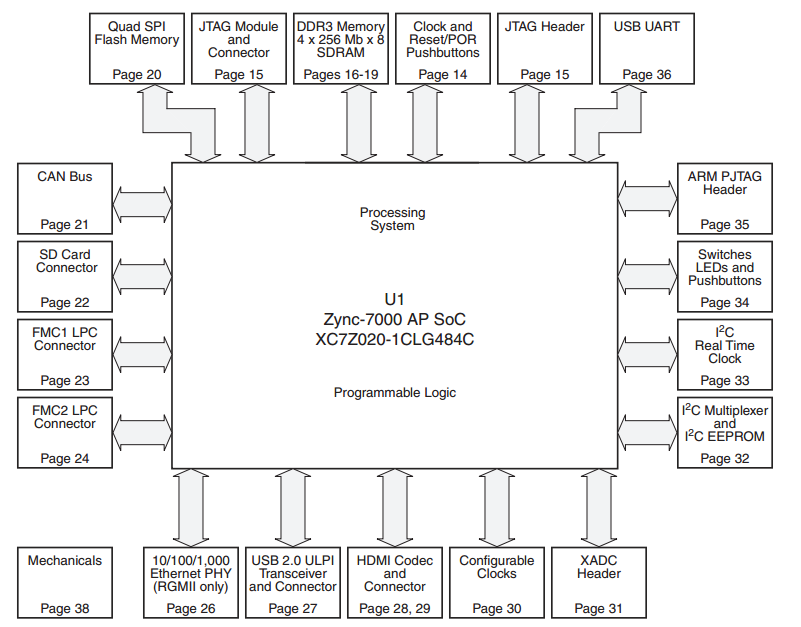


图 6 ZC702结构框图

# 三、AD9361和ZC702之间的数据通路

ADI公司提供了基于ZC702的硬件（PL侧）和软件（PS侧）设计，作为AD9361和Zynq之间的连接和使用的基础。在其提供的硬件设计的基础上，AD9361和ZC702之间的数据通路如所示：

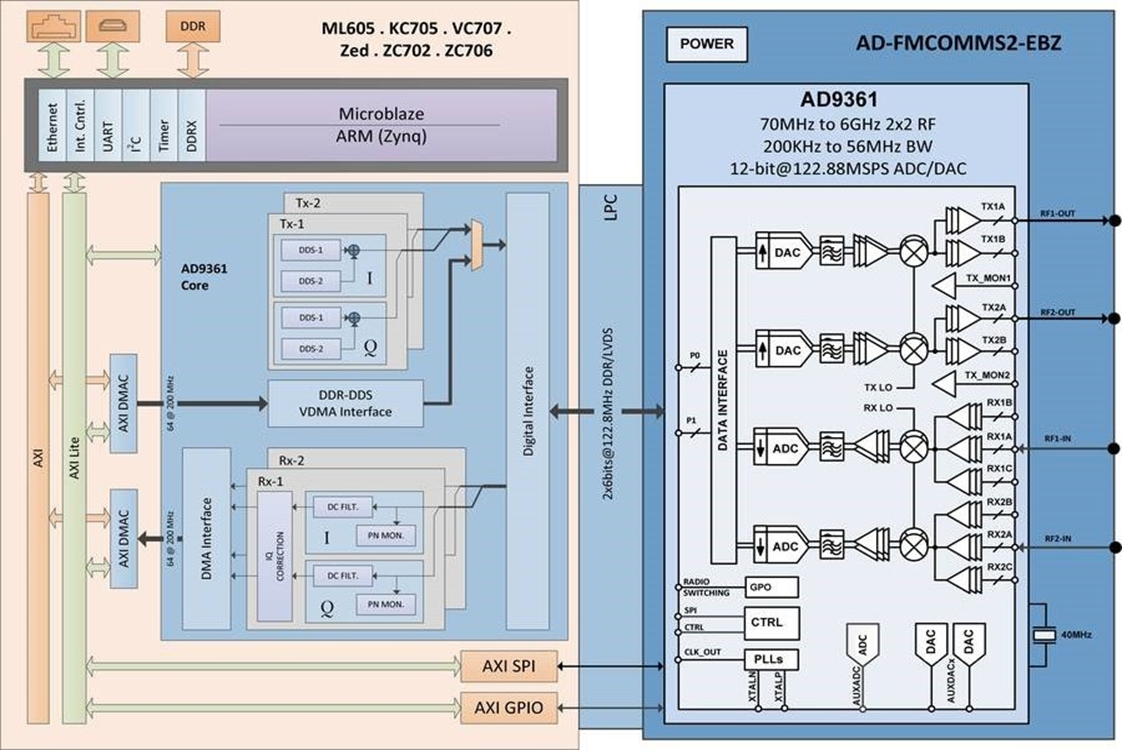


图 7 AD9361和ZC702之间数据通路

右边为AD9361部分，左边为ZC702部分；二者通过FMC接口传输数据。在ZC702部分的蓝色区域为zynq的PL部分，命名为AD9361 Core；上方为ARM部分，其余为接口部分，为PS和PL及AD9361之间建立数据通道。

ADI公司也提供了代码用于配置AD9361的参数和一些API用于数据的传输和接收；

与DDR进行传输数据的传输方式为DMA传输方式：在DMAC（DMA控制器）的操作下，数据直接由源地址传输到目的地址，（DDR<->AD9361）不需CPU的干预，因此可以极大的提高了CPU的效率。

注意：如果使用PL侧来产生用户数据而是直接通过PS侧来产生数据并发送的话，用户数据应该使用DMA传输方式和DDR。

在PFGA部分中DDS（直接数字式频率合成器）产生的信号数据和从AD9361接收到的信号数据都需要进行IQ调制和解调后，通过DMA方式传输到DDR中存储，这样方便ARM读取数据，并进行处理。

以上的数据通路是通过硬件设计来实现的，若想真正配置和使用AD9361，还需通过运行在ARM上的软件部分。

# 四、AD9361参考设计说明（PL侧硬件部分）

## 4.1 IP核的概念

IP（Intellectual Property）内核模块是一种预先设计好的甚至已经过验证的具有某种确定功能的集成电路、器件或部件。它有几种不同形式。IP内核模块有行为 （behavior）、结构（structure）和物理（physical）3级不同程度的设计，对应有主要描述功能行为的“软IP内核(soft IP core)”、完成结构描述的“固IP内核(firm IP core)”和基于物理描述并经过工艺验证的“硬IP内核(hard IP core)”3个层次。这相当于集成电路（器件或部件）的毛坯、半成品和成品的设计技术。

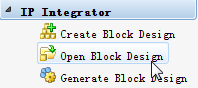
软核是用VHDL等硬件描述语言描述的功能块，但是并不涉及用什么具体电路元件实现这些功能。软IP通常是以硬件描述 语言HDL源文件的形势出现，应用开发过程与普通的HDL设计也十分相似，只是所需的开发硬软件环境比较昂贵。软IP的设计周期短，设计投入少。由于不涉 及物理实现，为后续设计留有很大的发挥空间，增大了IP的灵活性和适应性。其主要缺点是在一定程度上使后续工序无法适应整体设计，从而需要一定程度的软 IP修正，在性能上也不可能获得全面的优化。

因此，在AD9361参考设计中给出的IP核全为软IP内核

## 4.2 硬件设计

打开Vivado，打开…\hdl-hdl\_2014\_r2\projects\fmcomms2\zc702路径下的AD9361的Vivado工程文件fmcomms2\_zc702.xpr

点击右侧导航栏里的IP Integrator—Open Block Design



打开ADI公司给的AD9361的参考设计IP核框图，每个蓝色方块即为硬件部分的各个IP核，如图 8所示的部分（可放大观看），该部分为AD9361 Core和DMA控制器部分（其余可在Vivado中查看）。

各IP核的源Verilog HDL代码在hdl-hdl\_2014\_r2\library中，这些设计就是图 7中左半部分的设计

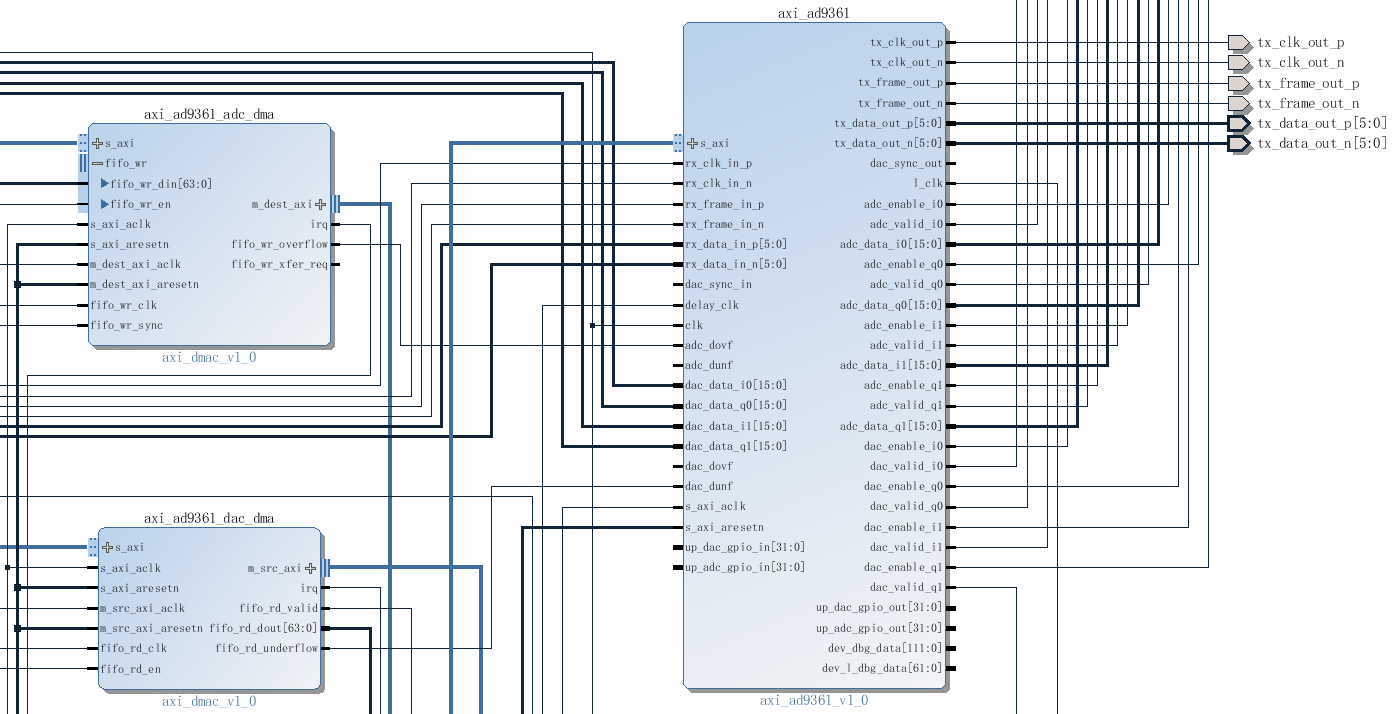


图 8 AD9361参考设计IP核框图（DMA和AD9361部分）

# 五、AD9361 参考设计说明（PS侧软件部分）

## 5.1 AD9361 no-OS Software 概述

AD9361 no-OS Software是ADI公司提供的AD9361的软件部分，运行在CPU（也就是Zynq的ARM）中，该程序为裸机程序（即无操作系统的程序），可以AD9361各个参数进行配置，对PL中的一些寄存器进行读写，控制发送数据源，控制DMAC（DMA控制器）对发送和接收的数据进行传输，从而实现AD9361的基本功能：对数据的接收、处理和发送。整个程序是使用C语言来完成的。

## 5.2 AD9361 no-OS Software 顶层目录说明

在ADI官网可以下载到no-OS-Software的源码，下载到的源码包含了很多ADI的收发器， 表 1列出的是AD9361的no-OS-Software中的文件和文件夹目录结构

|  |  |  |
| --- | --- | --- |
| **目录** | **子文件** | **解释说明** |
| console\_commands | command.h 、command.c | 包含用于控制AD9361的命令行的文件 |
| console.h、console.c | 包含用于命令行操作、显示等的与平台串口相关的文件 |
| platform\_altera |  | Altera平台的相关文件（使用Xilinx平台无需该文件） |
| platform\_generic |  | 通用平台的相关文件（使用Xilinx平台无需该文件） |
| platform\_linux |  | Linux平台相关的文件（使用Xilinx平台无需该文件） |
| platform\_xilinx | adc\_core.h、adc\_core.c | 模数转换模块控制文件，包括模块的初始化和数据传输等 |
| dac\_core.h、dac\_core.c | 模数转换模块控制文件，包括模块的初始化和数据传输等 |
| Platform.c、platform.h | Xilinx平台一些驱动文件 |
| parameters.h | 以上文件所用到的参数的宏定义文件 |
| ad9361.c |  | AD9361的驱动文件，比如增益控制函数等 |
| ad9361.h |
| ad9361\_api.c |  | AD9361应用编程接口驱动文件，比如AD9361的初始化函数 |
| ad9361\_api.h |
| common.h |  | 通用驱动文件，包含时钟结构体和通用宏定义 |
| config.h |  | AD9361和 AD9361 API的配置文件 |
| main.c |  | 整个软件部分的main函数文件 |
| util.c |  | util驱动文件 |
| util.h |  |

表 1 AD9361 no-OS-Software源码目录结构

## 5.3 main.c文件

main.c文件是main函数所在文件，是整个程序的入口。

main.c文件的开头是需要条件编译的头文件和宏定义。

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*Include Files \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/9*

**#include** *"config.h"*

**#include** *"ad9361\_api.h"*

**#include** *"parameters.h"*

**#include** *"platform.h"*

**#ifdef** CONSOLE\_COMMANDS

**#include** *"command.h"*

**#include** *"console.h"*

**#endif**

**#ifdef** XILINX\_PLATFORM

**#include** **<**xil\_cache**.**h**>**

**#endif**

**#if** **defined** XILINX\_PLATFORM **||** **defined** LINUX\_PLATFORM

**#include** *"adc\_core.h"*

**#include** *"dac\_core.h"*

**#include** *"adc\_interrupt.h"*

**#include** *"SD\_card.h"*

**#endif**

因此，在使用时需要根据情况在程序最开始对一些参数进行宏定义，

使用ZC702需要添加语句：

#define XILINX\_PLATFORM

如需使用命令行控制AD9361，需要添加语句：

#define CONSOLE\_COMMANDS

如需使用ADC的数据捕获功能，需要添加语句：

#define CAPTURE\_SCRIPT

然后是命令行函数所用到的一些变量的定义、对AD9361初始化所需要参数的变量定义和AD9361接收和发射端滤波器的定义。

AD9361\_InitParam default\_init\_param **=** **{**

*/\* Identification number \*/*

**0,** *//id\_no;*

*/\* Reference Clock \*/*

**40000000UL,** *//reference\_clk\_rate*

*/\* Base Configuration \*/*

**0,** *//two\_rx\_two\_tx\_mode\_enable \*\*\* adi,2rx-2tx-mode-enable*

**1,** *//one\_rx\_one\_tx\_mode\_use\_rx\_num \*\*\* adi,1rx-1tx-mode-use-rx-num*

**1,** *//one\_rx\_one\_tx\_mode\_use\_tx\_num \*\*\* adi,1rx-1tx-mode-use-tx-num*

**1,** *//frequency\_division\_duplex\_mode\_enable \*\*\* adi,frequency-division-duplex-mode-enable*

**0,** *//frequency\_division\_duplex\_independent\_mode\_enable \*\*\* adi,frequency-division-duplex-independent-mode-enable*

**0,** *//tdd\_use\_dual\_synth\_mode\_enable \*\*\* adi,tdd-use-dual-synth-mode-enable*

**0,** *//tdd\_skip\_vco\_cal\_enable \*\*\* adi,tdd-skip-vco-cal-enable*

**0,** *//tx\_fastlock\_delay\_ns \*\*\* adi,tx-fastlock-delay-ns*

**0,** *//rx\_fastlock\_delay\_ns \*\*\* adi,rx-fastlock-delay-ns*

**0,** *//rx\_fastlock\_pincontrol\_enable \*\*\* adi,rx-fastlock-pincontrol-enable*

**0,** *//tx\_fastlock\_pincontrol\_enable \*\*\* adi,tx-fastlock-pincontrol-enable*

**0,** *//external\_rx\_lo\_enable \*\*\* adi,external-rx-lo-enable*

**0,** *//external\_tx\_lo\_enable \*\*\* adi,external-tx-lo-enable*

**5,** *//dc\_offset\_tracking\_update\_event\_mask \*\*\* adi,dc-offset-tracking-update-event-mask*

**6,** *//dc\_offset\_attenuation\_high\_range \*\*\* adi,dc-offset-attenuation-high-range*

之后的部分是整个软件部分的主函数，主函数的程序流程图如图 6所示（默认定义了XILINX\_PLATFORM常量）：<main流程图.vsdx>



图 9 main函数流程图

DAC模块初始化使用的函数为dac\_init函数

ADC数据捕获使用到的函数为adc\_capture函数

这两个函数是控制数据传输的主要函数，下面的章节将会详细介绍这两个函数。

## 5.4 dac\_init函数

dac\_init为DAC模块初始化函数，也负责DMA传输部分，将DDR中的数据送给AD9361。

dac\_init函数的函数声明为：



第一个参数 struct ad9361\_rf\_phy \*phy 为指向AD9361的射频设备结构体的指针。

第二个参数 uint8\_t data\_sel为 需要发送的数据源的选择：

**enum** dds\_data\_select

**{**

DATA\_SEL\_DDS**,**

DATA\_SEL\_SED**,**

DATA\_SEL\_DMA**,**

DATA\_SEL\_ZERO**,** */\* OUTPUT 0 \*/*

DATA\_SEL\_PN7**,**

DATA\_SEL\_PN15**,**

DATA\_SEL\_PN23**,**

DATA\_SEL\_PN31**,**

DATA\_SEL\_LB**,** */\* loopback data (ADC) \*/*

DATA\_SEL\_PNXX**,** */\* (Device specific) \*/*

USER\_DATA**,**

**};**

0表示发送DDS生成的信号；

2表示通过DMA发送DDR中的信号数据，该数据在dac\_core.c开头定义；

3表示发送全0信号；

4-7表示发送随机数信号；

8表示发送从ADC中接收到的数据信号；

9表示发送选定设备的信号；

10为用户数据（需要在函数中添加代码）

第三个参数为DMA设置的标准位，0表示设置；1表示不设置。

下图为dac\_init函数的流程图：<dac_init流程图.vsdx>

图 10 dac\_init函数流程图

其中需要重点关注配置DAC的DMAC的部分（橙色部分），配置DMAC的流程图如图 8所示：



图 11 dac DMAC部分流程图

源代码如下:

dac\_dma\_write**(**AXI\_DMAC\_REG\_CTRL**,** **0);***//初始化DMA*

dac\_dma\_write**(**AXI\_DMAC\_REG\_CTRL**,** AXI\_DMAC\_CTRL\_ENABLE**);***//DMA通道使能*

dac\_dma\_write**(**AXI\_DMAC\_REG\_SRC\_ADDRESS**,** DAC\_DDR\_BASEADDR**);***//设置DMA的传输源地址为DDR的基地址*

dac\_dma\_write**(**AXI\_DMAC\_REG\_SRC\_STRIDE**,** **0x0);**

dac\_dma\_write**(**AXI\_DMAC\_REG\_X\_LENGTH**,** length **-** **1);***//Number of bytes to transfer - 1*

dac\_dma\_write**(**AXI\_DMAC\_REG\_Y\_LENGTH**,** **0x0);** *//Number of rows to transfer - 1*

dac\_dma\_write**(**AXI\_DMAC\_REG\_START\_TRANSFER**,** **0x1);** *//向传输队列中加入传输请求*

dac\_dma\_write函数的作用是向选定地址（第一个参数）中写入数值（第二个参数）

以上这些宏定义都是DMAC的寄存器地址：

AXI\_DMAC\_REG\_CTRL：DMAC总控制寄存器，两位00表示关闭DMA通道；01表示使能DMA通道；

AXI\_DMAC\_REG\_SRC\_ADDRESS：传输源地址寄存器，内容为DMA传输的源地址（即数据所在地址）；

AXI\_DMAC\_REG\_SRC\_STRIDE：传输每行的字节数；

AXI\_DMAC\_REG\_X\_LENGTH：传输数据的总字节数；

AXI\_DMAC\_REG\_Y\_LENGTH：传输数据的行数；

AXI\_DMAC\_REG\_START\_TRANSFER：传输开始寄存器，写入1会将新的传输加入传输队列；

关于这些寄存器和其完整的说明在[HDL Core RegMap.pdf](../../AD9361/HDL%20Core%20RegMap.pdf)

DMA传输支持2维传输（即按行列传输），但是目前只需要一维传输，因此，在源代码里向AXI\_DMAC\_REG\_SRC\_ADDRESS和AXI\_DMAC\_REG\_Y\_LENGTH写入0表示只使用一维传输。

在传输用户数据时，也应该在将用户数据进行调制后，参照以上代码，使用DMA方式发送数据

## 5.5 adc\_capture函数

adc\_capture函数为数据捕获函数，也负责DMA传输数据到 DDR

adc\_init函数的函数声明为：



第一个参数size为要捕获的数据量（个）；

第二个参数为start\_adress存储捕获数据的目的地址。

图 9为dac\_init函数的流程图：



图 12 adc\_captur函数流程图

源代码如下：

int32\_t adc\_capture**(**uint32\_t size**,** uint32\_t start\_address**)**

**{**

uint32\_t reg\_val**;**

**if(**adc\_st**.**rx2tx2**)**

**{**

length **=** **(**size **\*** **8);**

**}**

**else**

**{**

length **=** **(**size **\*** **4);**

**}**

上图的源码为adc\_capture函数的第一部分——数据单位转换部分，adc\_capture函数的第一个参数size为用户想要捕获到的数据量，单位是“个”，但是在DMAC的很多寄存器中，比如AXI\_DMAC\_REG\_X\_LENGTH寄存器，其中的数值为传输的数据的总字节数，单位为“字节”，因此需要将size单位转换为“字节”。如果打开了双通道，那么捕获的数据数据会占用双倍的存储空间。

adc\_dma\_write**(**AXI\_DMAC\_REG\_CTRL**,** **0x0);** *//初始化DMA通道*

adc\_dma\_write**(**AXI\_DMAC\_REG\_CTRL**,** AXI\_DMAC\_CTRL\_ENABLE**);***//DMA通道使能*

adc\_dma\_write**(**AXI\_DMAC\_REG\_IRQ\_MASK**,** **0x0);***//取消屏蔽.*

*//adc\_dma\_read(AXI\_DMAC\_REG\_TRANSFER\_ID, &transfer\_id);//读取下一个传输的ID号(5位)*

adc\_dma\_read**(**AXI\_DMAC\_REG\_IRQ\_PENDING**,** **&**reg\_val**);** */\*读取中断状态:一个传输完成后 END\_OF\_TRANSFER 即 [1]位 置 1，*

*一个传输加入队列后 START\_OF\_TRANSFER 即 [0]位 置 1 \*/*

adc\_dma\_write**(**AXI\_DMAC\_REG\_IRQ\_PENDING**,** reg\_val**);***//写入中断状态寄存器,使中断寄存器初始化*

adc\_dma\_write**(**AXI\_DMAC\_REG\_DEST\_STRIDE**,** **0x0);***//设置目的地址中从一行的开始和下一行之间的字节数*

adc\_dma\_write**(**AXI\_DMAC\_REG\_X\_LENGTH**,** length **-** **1);***//传输的字节数*

adc\_dma\_write**(**AXI\_DMAC\_REG\_Y\_LENGTH**,** **0x0);***//传输的行数*

adc\_dma\_write**(**AXI\_DMAC\_REG\_DEST\_ADDRESS**,** start\_address**);** *//设置传输的目的地址(destination address)*

adc\_dma\_write**(**AXI\_DMAC\_REG\_START\_TRANSFER**,** **0x1);***//加入传输队列*

上图为adc\_capture函数的第二部分——DMA配置部分，其中与ADC模块相比不同的寄存器为：

AXI\_DMAC\_REG\_IRQ\_MASK：中断屏蔽寄存器，[1]位为EOT（End Of Transfer）IRQ，[0]位为SOT（Start Of Transfer） IRQ，哪一位置1，就表示那一位的中断请求被屏蔽；

AXI\_DMAC\_REG\_IRQ\_PENDING：读取中断状态:一个传输完成后 END\_OF\_TRANSFER 即 [1] 位 置 1，一个传输加入队列后 START\_OF\_TRANSFER 即 [0] 位 置 1 \*/

AXI\_DMAC\_REG\_TRANSFER\_ID：该寄存器的数值为下一次传输的ID号。

*//Wait until the new transfer is queued.*

**do**

**{**

adc\_dma\_read**(**AXI\_DMAC\_REG\_START\_TRANSFER**,** **&**reg\_val**);**

**}**

**while(**reg\_val **==** **1);\*/**

*// Wait until the current transfer is completed.*

**do**

**{**

adc\_dma\_read**(**AXI\_DMAC\_REG\_IRQ\_PENDING**,** **&**reg\_val**);**

**}**

**while(1);***//reg\_val !=0011b\*/*

*//Wait until the transfer with the ID transfer\_id is completed.*

**do**

**{**

adc\_dma\_read**(**AXI\_DMAC\_REG\_TRANSFER\_DONE**,** **&**reg\_val**);** *//读取传输完成的ID号*

**}**

**while((**reg\_val **&** **(1** **<<** transfer\_id**))** **!=** **(1** **<<** transfer\_id**));**

上图为上图为adc\_capture函数的第三部分——判断与等待

1. 等待，直到一个新的传输加入传输队列

读取AXI\_DMAC\_REG\_START\_TRANSFER寄存器的值，值为1时循环，值为0时跳出循环。

之前已经向AXI\_DMAC\_REG\_START\_TRANSFER写入了1，在这时判断AXI\_DMAC\_REG\_START\_TRANSFER的值，若是1，表示新的传输仍然在排队，若是0，表示新的传输已经开始。

1. 等待，直到目前的传输完成。

读取AXI\_DMAC\_REG\_IRQ\_PENDING的值，当传输进行时，AXI\_DMAC\_REG\_IRQ\_PENDING的[0]位SOT位始终为1，当传输完成时，[1]为EOT位由0置为1，之后两位都会被清0。因此，当AXI\_DMAC\_REG\_IRQ\_PENDING的值为3时，表示传输完成。

1. 等待，直到ID为transfer\_id的传输完成

这一步是为了验证之前设置的传输已经完成。