一种基于 HDL 的 TOTP 实现

本文描述了一种基于 verilog HDL 和 FPGA 的 TOTP（基于时间的一次性密码）算法的实现。

崔天一 PB14000202

目录

1 引入

近年来，随着互联网的普及以及人们对于信息安全的重视，普通的基于密码的用户身份认证体系受到了极大的挑战。有将近 4 成网民有密码被盗的经历。[1]在网络上进行金融操作（转账、购物等）具有极大的风险性。

最近，为了应对基于账号密码验证不安全的风险，各大商业银行新增加了基于手机短信、U 盾或动态密码器的二次验证。当用户需要进行高风险交易时，需要进行第二次验证。

基于短信的二次验证方法为网站向预先登记的手机发送含有 6 位验证码的短信，用户将 6 位验证码输入至网页中进行验证，从而证明该用户拥有该手机号，即该用户为本人。

基于 U 盾的验证方法是指用户将提前获得的 U 盾插入至计算机的 USB 接口中，通过验证用户确实持有令牌来确认用户为其本人。

基于电子密码器的验证方法是指银行提前如下图所示，为一银行颁发的电子密码器。

当用户申请网络银行时，银行会为用户颁发一个预置了**密钥**的，**已加电**的，内置有已与银行服务器同步的**时钟**的电子密码器。当用户打开显示功能后，该电子密码器会根据密钥和当前时钟计算出一个 6 位验证码。服务器同时存有对应的密钥，也可以计算出相同的 6 位验证码。服务器对用户的输入进行检查，从而判断用户是否真正持有电子密码器，从而判断用户是否为其本人。

除国内各大银行外，腾讯、阿里、百度等也同样开始采用 2 步验证的方法，在进行敏感操作前，需验证手机验证码，但 U 盾由于操作不便等原因，这些网站使用智能手机客户端实现了电子密码器的功能，用户安装客户端后，网站使用 2 维码与客户端交换共享密钥，之后客户端与服务器进行同步，从而实现电子密码器的所有功能。

然而以上基于软件客户端的电子密码器存在着需要用户持有智能手机的限制，不能被广泛应用。市场上目前所有的基于硬件的电子密码器算法又基本都没有公开，这让广大缺乏硬件实力的企业难以得到该种技术。为此，本文实现了一种基于 verilog HDL 的电子密码器。

2 功能

版内存有当前系统时间和密钥，该板会使用 8LED 和 4 个 7 段数码管显示 6 位验证码，高 2 位使用 LED 显示，使用 8421bcd 码进行表示，[led7:led4]显示验证码最高位，[led3:led0]显示验证码次高位，4 个数码管显示验证码的第 4 位。

当板内时钟与 Google 服务器时钟不一样时，提供时钟校正功能。可将正确的时钟置入板内。

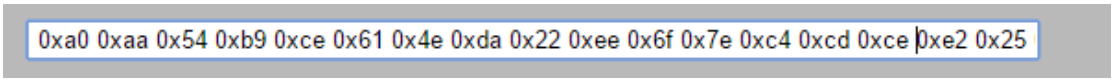
3 操作方法

获得密钥并刷入 FPGA

1. 访问 https://accounts.google.com/b/0/SmsAuthSettings#devices 获得 2 步验证 2 维码，如下图所示。

2. 使用二维码扫描器进行扫码，如下图所示：

3. 可以看到上图中的 secret 参数，将 secret 参数送入一些在线 base32 decode 网站（如 http://tomeko.net/online_tools/base32.php?lang=en）进行解码，下图为解码后的结果

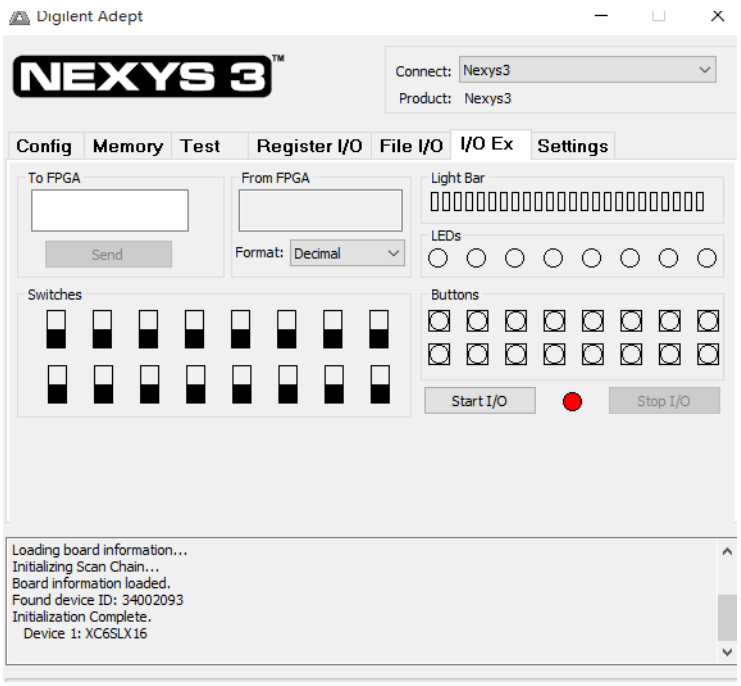


4. 将上述结果写入 main.v 的 secret 参数,进行编译、下载至板子上

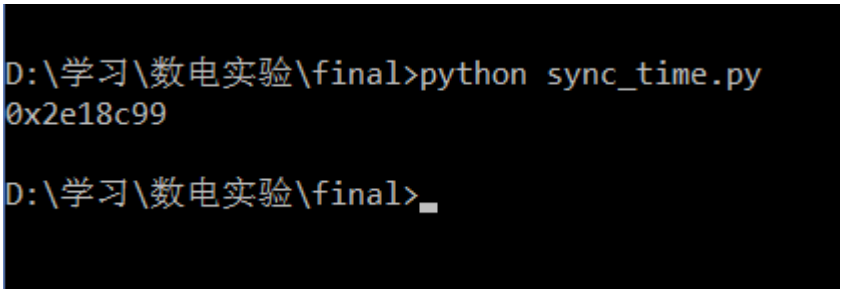
将板载时钟与服务器时钟同步

1 将电脑时钟与 ntp 服务器进行时钟同步

2 打开 Diglent Adept 软件，转至 I/O EX 界面，打开 start 按钮。



3 使用附件中的 sync_time.py 文件获得当前时间戳，如下图所示



4 将屏幕上显示的数字+1 填写至 2 中 to FPGA 输入框中，点击 Send 按钮，接着打开板子最靠右侧的按键（sw0）,按下正中间的 btn(B8)，从而将时钟写入板子。

获取验证码

获取验证码的方法如下图所示，系统使用 8 个 led 和 4 个 7 段数码管显示当前的动态验证码，使用 8421 码作为编码方法。图片上的 6 位验证码为 456274。

4 算法流程

该算法兼容于 RFC 6238[2]文档，即实现了一种基于时间的动态验证码生成功能，使用与 google authenticator 相同的计算方式，因此，该设计可以用来进行所有 Google 产品的 2 次验证。请详见上文的案例以及附件中的视频。

一些名词的约定

- Key 用户从服务器获得的解码后的标志用户唯一身份的密钥,使用大端字节序
- Ipad 使用 0x36 填充的长为 512 位的字符串
- Opad 使用 0x5C 填充的长为 512 位的字符串
- Timestamp 8 字节，64 位长的从协调世界时 1970 年 1 月 1 日零时零分零秒至今所经过的秒数/30，使用小端序存储
- ||(0) 使用 0 对串进行填充
- Hash(X) 使用 SHA1 哈希算法对 X 操作后所得到的值
- A||B 将二进制串 A 和 B 进行连接，相当于+
- A ⊕ B 将 A 与 B 进行亦或，如果长度不等，将短的补 0

密钥 key 的生成

密钥为一个长度小于 512 位的 0/1 二进制串，为了使其可以在屏幕上正常显示，使用 base32[3]对其进行编码并下发给用户。用户需要将其在本机上进行解码后作为参数输入 verilog HDL 代码中。

如下为一个示例 base32 编码后的密钥：

caaaaaaaaaaaaaaa

经 base32 解码后可以得到密钥为：

0x10000000000000000000

计算 HMAC 值[4]

HMAC 是一种对消息进行验证的哈希算法，基于 rfc2104 实现。其流程如下所述：

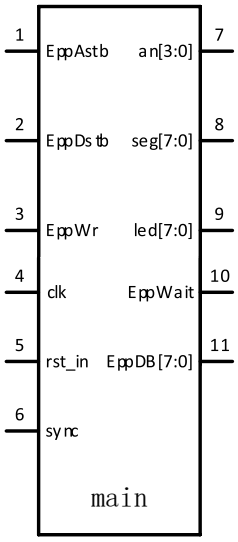
（1） 计算 key ⊕ Ipad
（2） 计算 key ⊕ Ipad || timestamp
（3） 计算 Hash(key ⊕ Ipad || timestamp)
（4） 计算 key ⊕ Opad
（5） 计算(key ⊕ Opad) || Hash(key ⊕ Ipad || timestamp)
（6） 计算 Hash((key ⊕ Opad) || Hash(key ⊕ Ipad || timestamp))

对得到的 HASH 进行采样

上步（6）得到的值为 160 位，20 字节数值，假设该 20 字节为 a[i] (i=0~19)取其最后一个字节的低 4 为（0-15）作为索引号，对其取 a[a[0] & 0x0F]:a[a[0] & 0x0F + 4]得到一个 32 位有符号 2 进制数，将其对 1000000 取模，并转换为 8421 码，从而获得最后的验证码。

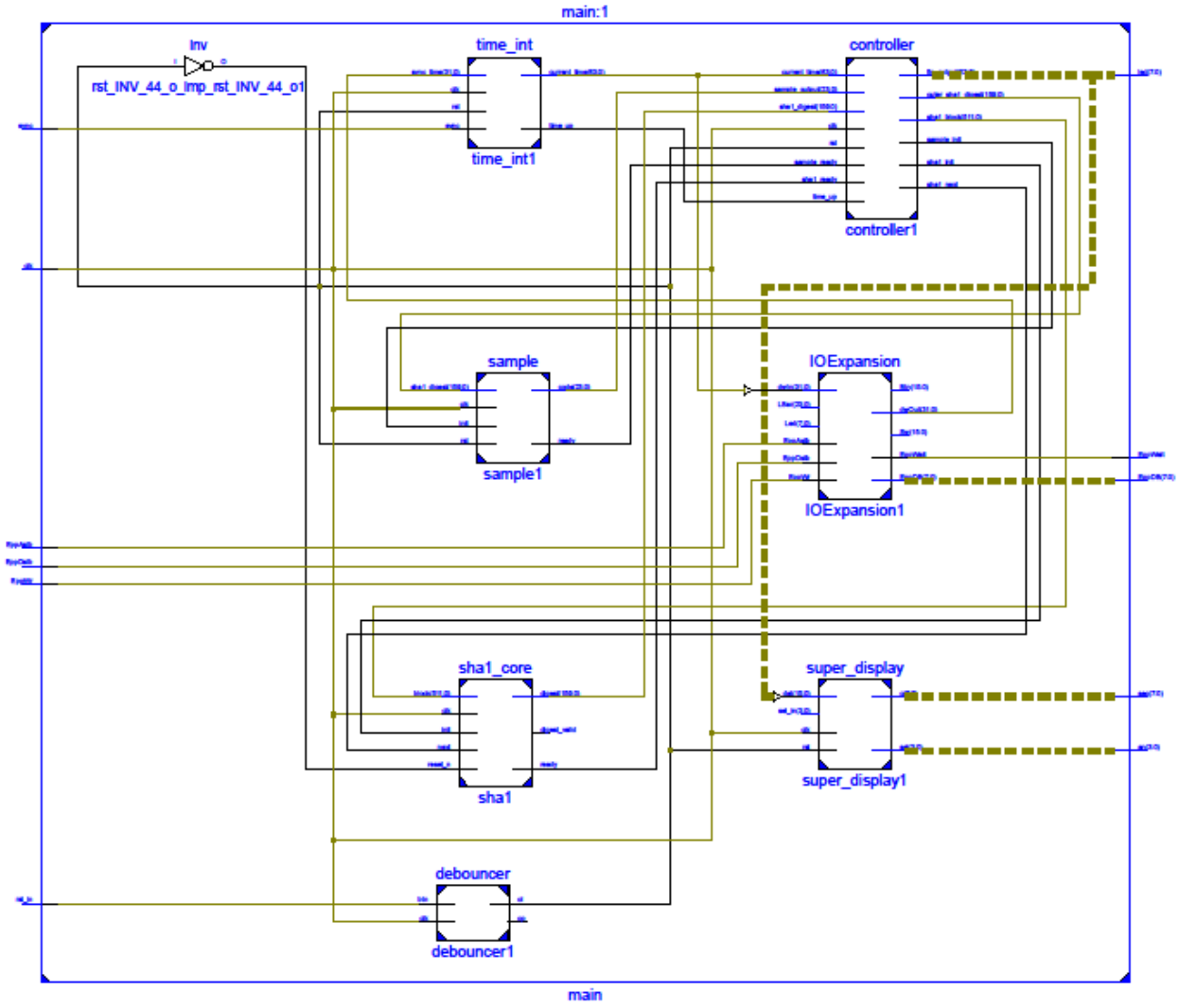
5. 数据通路和模块设计

顶层设计及数据通路

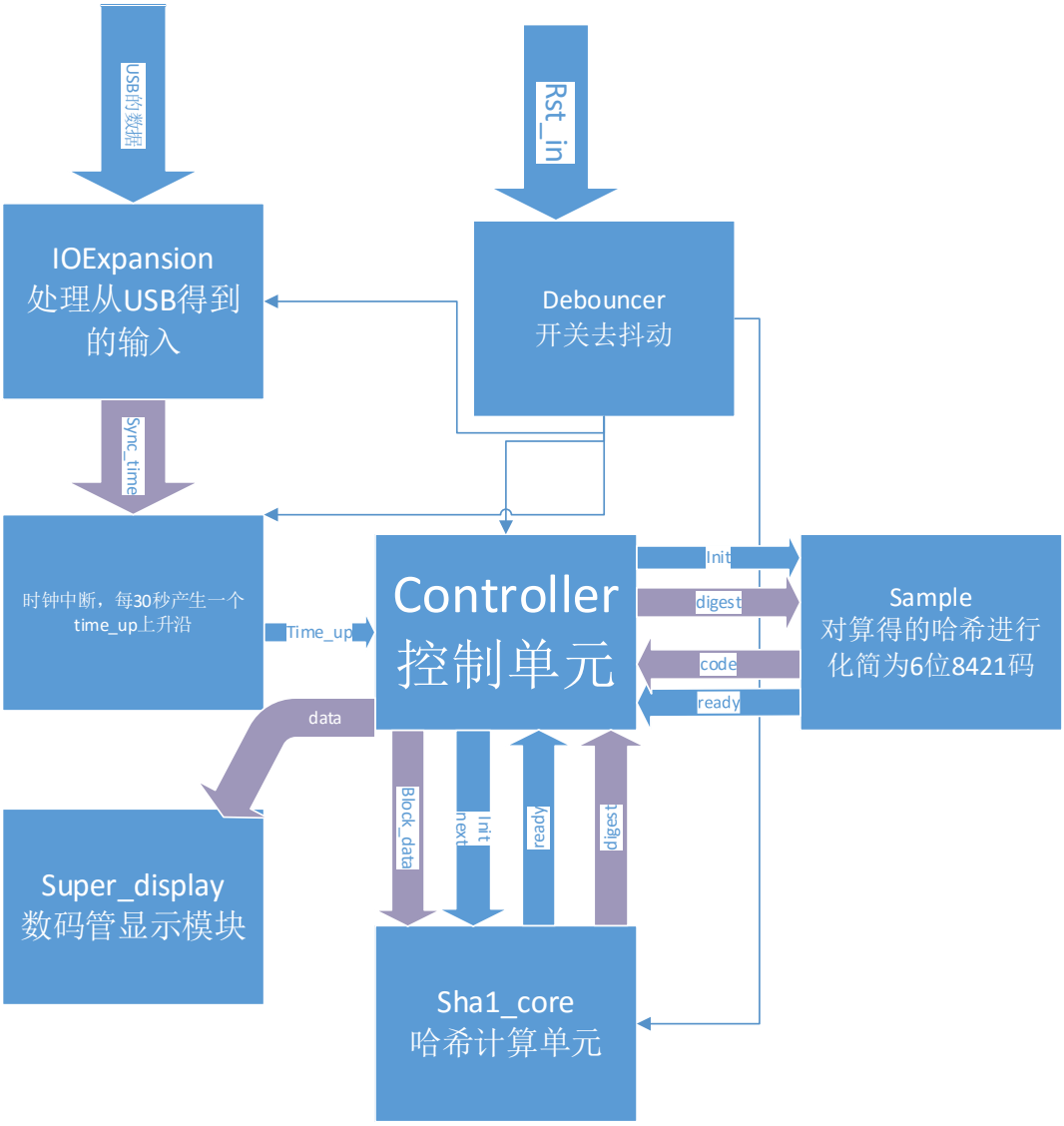


左图为顶层模块 EppAstb EppDstb,EppWr,EppWait,EppDB 为通过 USB 同步时间戳所使用的端口，clk 为时钟输入,rst_in 为复位信号，高电平有效。Seg as 控制 7 段数码管，led 控制 led 灯。

数据通路如下图所示：


main:1
Inv
rst_INV_44_o_imp_rst_INV_44_o1
time_int
time_int1
controller
controller1
sample
sample1
IOExpansion
IOExpansion1
sha1_core
sha1
super_display
super_display1
debouncer
debouncer1
main

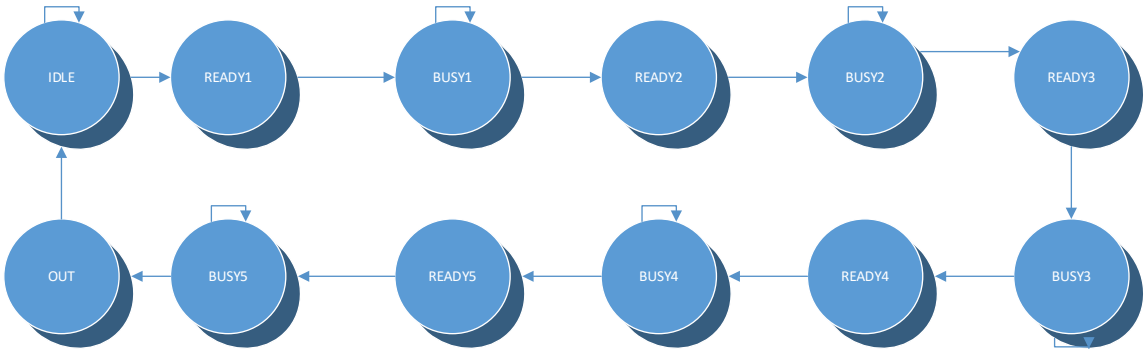
简化版本如下图所示：



控制器控制各个模块的状态，向各个模块发出请求，各个模块在计算完成后将 ready 信号置 1，控制器轮询等待直到各个模块完成工作。其中，细蓝色线为 rst 信号，紫色箭头为数据信号通路，蓝色箭头为控制信号通路。

各个模块的设计

控制器(Controller)：一个极其复杂的有限状态自动机，按照算法流程所述，依次控制各个模块的运行。状态机如下图所示：



采样模块（Sample）：执行算法中“对得到的 HASH 进行采样”的步骤。
时钟模块（time_int）:每 30s 产生一个信号上升沿，并输出当前的 Unix 时间戳/30 后的值
IO 扩展：从 USB 口读入当前时间

参考文献

[1] http://news3.xinhuanet.com/newscenter/2006-06/04/content_4643383.htm 新华网
[2] https://tools.ietf.org/html/rfc6238 TOTP: Time-Based One-Time Password Algorithm
[3] https://tools.ietf.org/html/rfc4648 The Base16, Base32, and Base64 Data Encodings
[4] https://tools.ietf.org/html/rfc2104 HMAC: Keyed-Hashing for Message Authentication