**USB digital domain chip specification**

**(Document Control: V.0.3)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Note** |
| 0.1 | 03/16/2021 | Bao Tran | Draft Release. |
| 0.2 | 03/18/2021 | Bao Tran | Updated otp controller fsm, APB bus, register contribution, rcm (combine system clock and interface clock) |
| 0.3 | 03/19/2021 | Bao Tran | Updated otp controller fsm, register contribution, rcm.  Implement DFT scan chain |
| 1.1 | 03/24/2021 | Bao Tran | Update otp controller with 1Kbits IP. Compatible with IP tef65gp128x8hd\_140b. |
|  | MM/DD/YYYY | Your Name | Changed something. (Big change) |

**Contents**

[**Contents** i](#_Toc66794069)

[**Figures** i](#_Toc66794070)

[**Tables** ii](#_Toc66794071)

[**Conventions** ii](#_Toc66794072)

[1 Introduction and Features 1](#_Toc66794073)

[1.1 Introduction 1](#_Toc66794074)

[1.2 Features 1](#_Toc66794075)

[2 Module Overview 1](#_Toc66794076)

[2.1 Chip Block Diagram 1](#_Toc66794077)

[2.2 Functionality 1](#_Toc66794078)

[2.2.1 OTP Controller 2](#_Toc66794079)

[2.2.2 I2C interface 8](#_Toc66794080)

[2.2.3 Hif\_idle signal 8](#_Toc66794081)

[2.2.4 Register bus 8](#_Toc66794082)

[2.2.5 Register MUX 8](#_Toc66794083)

[3 Requirement 8](#_Toc66794084)

**Figures**

[**Figure 1. Chip Block Diagram** 1](#_Toc67299753)

[**Figure 2. FSM of chip operation** 3](#_Toc67299754)

[**Figure 3. FSM of OTP controller** 3](#_Toc67299755)

[**Figure 4. Timing diagram of OTP in program operation** 4](#_Toc67299756)

[**Figure 5. FSM of otp controller in program operation** 4](#_Toc67299757)

[**Figure 6. A sample of otp controller in program operation** 5](#_Toc67299758)

[**Figure 7. Timing diagram of OTP in read operation** 5](#_Toc67299759)

[**Figure 8. FSM of otp controller in read operation** 6](#_Toc67299760)

[**Figure 9. A sample of otp controller in read operation** 6](#_Toc67299761)

[**Figure 10. FSM of generating run\_test\_mode bit** 7](#_Toc67299762)

[**Figure 11. Mechanism of transfer data from register to otp** 7](#_Toc67299763)

[**Figure 12. Shift register used to program data to OTP** 8](#_Toc67299764)

[**Figure 13. Mechanism of transfer data from otp to register** 8](#_Toc67299765)

[**Figure 14. Shift register used to store data from OTP** 8](#_Toc67299766)

[**Figure 15. Synchronize system reset with system clock** 9](#_Toc67299767)

[**Figure 16. Combination between system clock and interface clock to register clock** 9](#_Toc67299768)

[**Figure 17. Write transfer** 10](#_Toc67299769)

[**Figure 18. Read transfer** 10](#_Toc67299770)

[**Figure 19. FSM for APB master** 11](#_Toc67299771)

[**Figure 20. FSM for APB slave** 12](#_Toc67299772)

[**Figure 21. Connection between APB master and slave** 12](#_Toc67299773)

[**Figure 22. OTP distribution in usb chip** 13](#_Toc67299774)

[**Figure 23. Schematic of test en generator** 14](#_Toc67299775)

[**Figure 24. Schematic of soft reset generator** 14](#_Toc67299776)

[**Figure 25. Schematic of i2c\_clk generator** 14](#_Toc67299777)

[**Figure 26. Schematic of system clock generator** 15](#_Toc67299778)

[**Figure 27. Output to analog circuits generator schematic** 15](#_Toc67299779)

[**Figure 28. Implement sda/scan\_out IO Pad** 15](#_Toc67299780)

**Tables**

[**Table 2‑1 Interface Table** 2](#_Toc340650810)

[**Table 2‑2 Register Table** 3](#_Toc340650811)

**Conventions**

This document uses the following notational conventions:

|  |  |
| --- | --- |
| **fr\_mod\_sig** | Ports are shown in lowercase bold |
| *mod\_reg* | Italic indicates registers or wires |
| 6’h00 | The hexadecimal number with the same form defined in IEEE Std 1364-2001 |
| 6’b00\_0000 | The binary number with the same form defined in IEEE Std 1364-2001 |
| ! | NOT logical operator |
| & | AND logical operator |
| | | OR logical operator |
|  |  |
|  |  |
|  |  |

## Introduction and Features

### Introduction

### Features

* Communicate with host processor via I2C interface
* Load OTP values to the register file in the beginning (boot phase), or when there is a NVM load request in the test mode
* Program OTP in the test mode

## Module Overview

### Chip Block Diagram

The chip level block diagram that includes this module is shown as below.

 **Figure 1. Chip Block Diagram**

### Functionality

The OTP works only during the boost phase or in the test mode. The digital part also works only when users read/write data from/to register file.

After system reset, the “APB mux” connects the “OTP controller” to the “Register file”. Then we do the OTP loading to load data values from the OTP to the register file. These values are used to trim the analog circuits for instance: referent voltages, referent currents, oscillating frequencies… During this time, the “I2C interface” are disconnected from the “Register file” and no read/write transactions to the interfaces take effect. During the boot time, the “Reset and Clock Management” (RCM) use the fast system clock to provide the “OTP controller” and the “Register file”. RCM provides interface clock to the “I2C interface”.

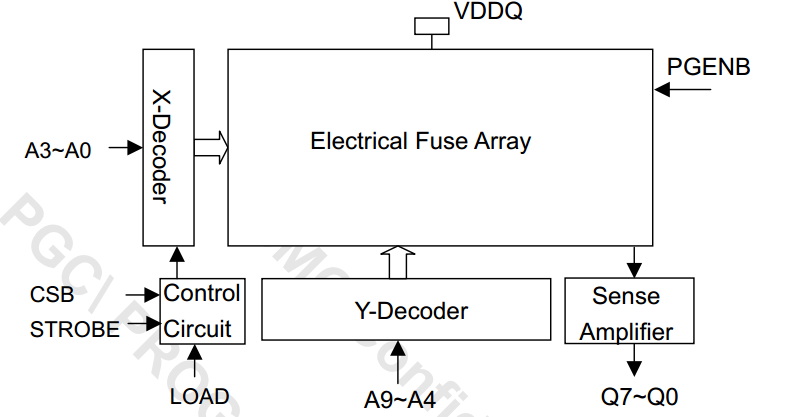
After “OTP controller” finish loading data, it sets bit “otp\_busy” to 0. When “otp\_busy” bit is set to 0 and not during I2C transaction (“i2c\_busy” = 0), the “APB mux” switches to connect the “I2C interface”, and the “RCM” uses the interface clock to provide the “Register file”. As a result, the “I2C interface” and the “Register file” use the common clock source for reading and writing. In addition, when there is no interface transaction (in most of time), the system clock is idle and there is no dynamic power consumption.

After OTP loading, the chip is in working phase normal mode and users can read/write value to the “Register file” via “I2C interface”. In the normal mode, users read/write values to the “Register file” to control the analog circuit. All configuring bits of the “Register file” are connected directly to the analog circuit. However, there are some exceptions. Some configuring bits have to take effect after finishing interface transaction. These bits include “soft\_reset”, “otp\_read\_n”, and “otp\_pgm”. If these bits take effect immediately, the interface transaction will be interrupted. It is possible to read/write multiple bytes in one transaction. We implement one counter inside the interface block to increase the register address automatically after reading/writing 1 byte. The counter saturates to the highest address value.

To enter the test mode, we write a passcode (“PHSGNX”) to register TEST0 in the register file in 6 consecutive i2c transactions. After that we can read/write to the test registers of the “Register file”. If we are not in the test mode, writing to test registers does not take effect.

#### eFuse memory

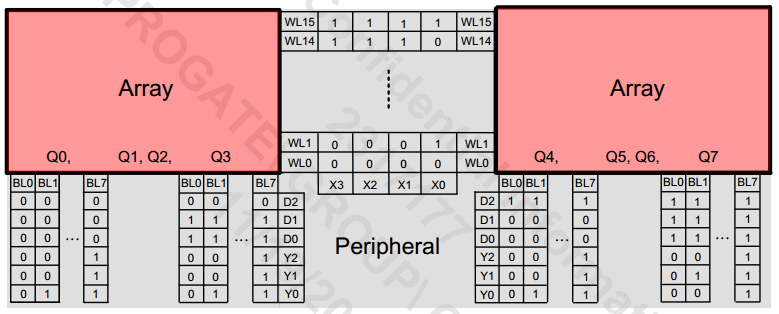
The block diagram of the eFuse memory is shown in the following figure.



**Figure 2. Block digram of eFuse memory**

Otp\_strobe is an input signal used to turn on the array for read or program access. Otp\_load is an input signal used to turn on sense amplifier and load data into latch. In the read mode, a 8-bits paraller will be read out by raising otp\_strobe high with a proper address selected. The otp address includes:

* A[3:0] is used as word line decode.
* A[6:4] is used as bit line decode.
* A[9:7] is used as order of bit decode.



**Figure 4. eFuse 128x8 bits distribution**

During programming mode, **any bit in the macro can be programmed in any order** by raising otp\_strobe high with a proper address selected.

During read mode, **a byte** (A[9:7] changes from 0 to 7) **can be read out** by raising otp\_strobe high with a proper address selected.

#### OTP Controller

##### Operating FSM

The FSM of the chip operation is shown in the following figure.



**Figure 5. FSM of chip operation**

When we reset usb chip, the FSM is in “idle” state. When “otp\_busy” is set to 1 and not during I2C transaction (“i2c\_busy” = 0), we move to “OTP\_controller” state. In this state, the “APB mux” switches to connect the “OTP controller” to program/load data to/from OTP. After OTP controller finish loading data, it sets “otp\_busy” to 0. Once there is no I2C transaction (“i2c\_busy” = 0), we move to “idle” state.

To read/write values to the “Register file” to control the analog circuit, we need to use i2c interface and finish otp controller operation. Therefore, once “i2c\_busy” is set 1 and “otp\_busy” is set to 0, the FSM is moved to “I2C” state. After completing an I2C transaction, “i2c\_busy” is set to 0 and we move to “idle” state.

##### Program commands

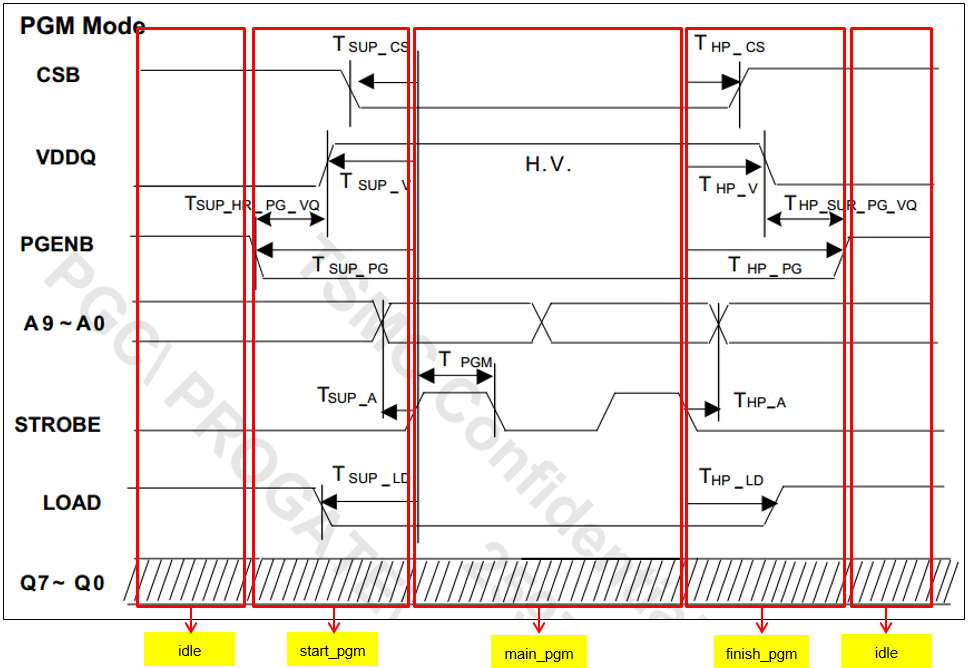


**Figure 6. FSM of OTP controller**

The FSM of the otp controller is shown in above figure. The FSM is designed based on 3 main operation:

* When **the usb chip is turned on**, system reset signal remains 0, the FSM starts at “start\_read” state to be ready for a read transaction to load otp data to register file. When system reset signal is released, otp controller start the read transaction. After completing first loading data, otp controller stays in “idle” state until next transaction.
* After loading data in the boot phase, once otp controller receives run\_test\_mode (access test mode) and i2c\_busy (no during transaction) and bit “otp\_prog” is set to 1, otp controller starts a **program transaction**.
* After loading data in the boot phase, once receiving run\_test\_mode (access test mode) and i2c\_busy (no during transaction) and bit “otp\_read\_n” is set to 0, otp controller starts a **read transaction**.

To describe the otp controller operation in detail, the FSM will be shared into two parts: one is to describe otp controller in program mode, and the other is to describre otp controller in read mode.



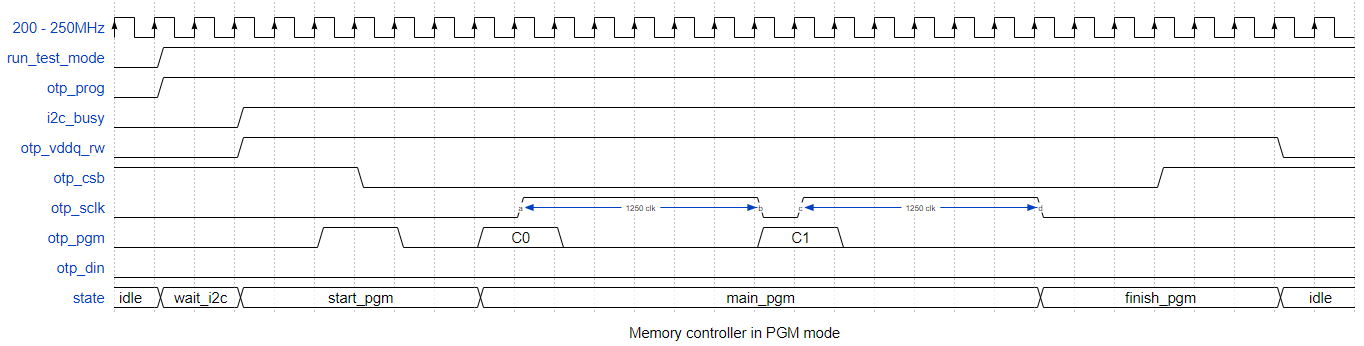
**Figure 7. Timing diagram of OTP in program operation**

Based on timing diagram of OTP in program operation (p.8 eFuse datasheet), the FSM of the otp controller in program operation is shown in the following figure.



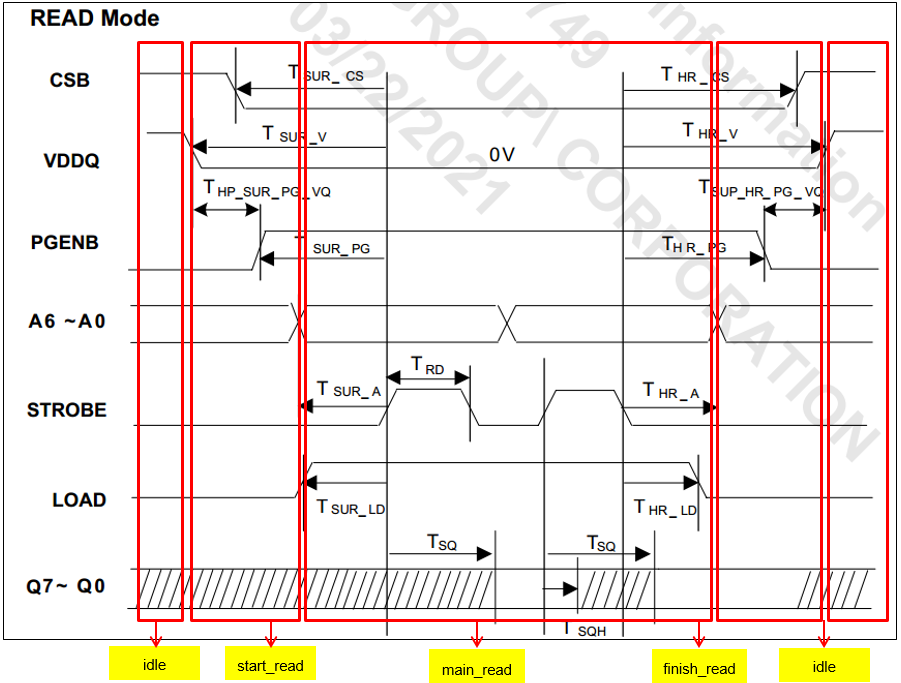
**Figure 8. FSM of otp controller in program operation**

As we can see in the figure, there are 5 states in the program FSM, includes idle, wait\_i2c, start\_pgm, main\_pgm and finish\_pgm. We always start a PGM/read operation with “idle” state. After the “run\_test\_mode” is set to 1, we move to “wait\_i2c” state. We only do pgm/read transaction to otp if there is no i2c transaction at the same time. Therefore, when receiving i2c\_busy = 0 and otp\_prog = 1, we move to “start\_pgm” state. In this state, otp controller need to generate main signals, such as otp\_vddq\_sw (used to turn on/off high voltate domain providing to OTP memory), otp\_csb, otp\_sclk, otp\_pgm, etc … Beginning “start\_pgm” state, the onset of programming mechanism starts at falling edges of otp\_csb while otp\_pgm is high and otp\_sclk is low. Also, otp\_vddq\_sw need to be set high before falling edge of otp\_csb. otp\_din should stay low for the entire programming cycle. We stay in this state in 44 cycles (to pass timing specification of THP\_CK). There is a timing counter (timing\_cnt) to counter the number of cycles. The counter runs in start\_pgm, main\_pgm and “finish\_pgm” state. In other state, its value is 0. After 44 cycles, we move to “main\_pgm” state. In “main\_pgm” state, for the subsequent fuse programming from bit 0 to the last bit (n-1), a specific fuse bit will be programmed if otp\_pgm is held high at the rising edge of otp\_sclk. Noted that we need 1250 clocks to complete 1 bit programming. After finishing programming data to the last bit of OTP, we set bit “otp\_lastbit” to 1 and move to “finish\_pgm” state. In the state, otp controller drives otp\_csb to high and then, after 3 clocks, otp\_vddq\_sw is also set to low. Figure 5 is an sample of otp controller in program operation.



**Figure 9. A sample of otp controller in program operation**

##### Read commands



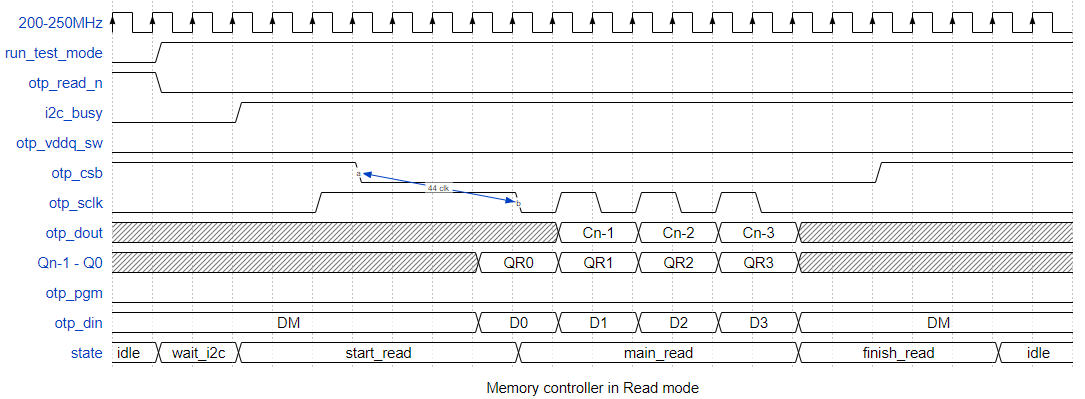
**Figure 10. Timing diagram of OTP in read operation**

Based on timing diagram of OTP in read operation (p.8 eFuse datasheet), the FSM of the otp controller in read operation is created and shown in the following figure.

As we can see in the figure, there are also 5 states in the read FSM, includes idle, wait\_i2c, start\_read, main\_read and finish\_read. After the “run\_test\_mode” is set to high, we move to “wait\_i2c” state. Then, when i2c\_busy and otp\_read\_n are low, we move to “start\_read” state. In the state, otp\_vddq\_sw is driven to low. To enter read mode, otp\_pgm and otp\_sclk must be set to low and high, respectively, at falling edges of otp\_csb. After the moment, all signals are kept stable during 44 cycles (to pass timing specification of THR\_CK). There is timing counter (timing\_cnt) to count the number of cycles. The counter runs in “start\_read”, “main\_read” and “finish\_read” state. In other state, its value is 0. Next we move to main\_read to perform load data from OTP and write to corresponding register. In the “main\_read” state, the OTP data are shifted out to otp\_dout at each falling edge of otp\_sclk in reverse order (from bit n-1 to bit 0). Hence, we need a counter to count down data in this state. The data load into register by otp\_din signal for soft repair are kept 0. When otp controller read the last data (bit 0), we set bit “otp\_lastbit” to 1 and move to “fnish\_read” state. In “finish\_read” state, otp controller drives otp\_csb to high. Then, the FSM goes to “idle” state to wait next transaction. Figure 8 is a sample of otp controller in read operation.



**Figure 11. FSM of otp controller in read operation**



**Figure 12. A sample of otp controller in read operation**

##### run\_test\_mode generation FSM

To avoid users program invalid data to OTP, we implement a special mechanism to access OTP controller by generating enable bit “run\_test\_mode”. This means that users can only access OTP controller in test mode and need to know the way to access this mode.



**Figure 13. FSM of generating run\_test\_mode bit**

To enter the test mode, we (or users) write a magic passcode (“PHSGNX”) to register TEST0 of the register file in 6 consecutive i2c transactions. After that we can read/write to the test registers of the “Register file”. If we are not in the test mode, writing to test registers does not take effect.

As we can see in the figure, there are 6 main states in the FSM: written\_P, written\_H, written\_S, written\_G, written\_N and written\_X. Besides, the FSM has “idle” and “run” state. In each main states, after every i2c transaction, we check if users write passcode correctly or not. If users write passcode correctly, bit “run\_test\_mode” will be set high and the usb chip is ready to use otp memory.

##### Load data from register file to OTP

To program or load data to/from otp, we divide the otp 1024 bits to different space. The otp consits of 16 word lines and 8 bit lines, while each bit line contains 1 byte data. Therefore, first word line stores data 16 first regsiters. The second word line data next 16 registers, so on… The following figure shows the otp distribution with corresponding register.



**Figure 14. Mechanism of transfer data from register to otp**

The above figure describes the way otp controller get value from register file and load these values to corresponding eFuse memory parts. Since OTP will be programmed from bit 0 to the last bit n-1, we need an addr up counter (addr\_cnt) to count up the number of address.

##### Program data from OTP to register file



**Figure 16. Mechanism of transfer data from otp to register**

The above figure describes the way otp controller get value from otp and write these values to corresponding register. Contrary to loading data from register file to otp, otp will load data from otp at each rising edge of otp\_strobe from addr 0 to addr 128. In this case, we need a an addr up counter (addr\_cnt) to count up the number of address.

##### Reset and Clock management

In the section, we present the clock and reset signals of the digital part of usb chip.

The main reset signal is the system reset “reset\_n” (or “rst\_n”). It is the Power on Reset (PoR) signal. In other words, when one of the power supplies is smaller than a threshold, then the signal “reset\_n” = 0. When all of the power supplies are higher than the threshold, we will release the reset signal (“reset\_n” = 1). In addition, to support soft reset, we also change the reset signal to ‘0’ when the config bit “soft\_reset” = 1.

We assume that during the system reset time (after turn on the power or after soft reset), there is no interface clock. This assumption is reasonable because we can state in the product datasheet that customers need to wait for a specific period after turn on the power or implement soft reset before read/write to the chip. In the rest of the chip, there is only system clock and slow clock. To avoid the meta-stability, we need to synchronize the reset with the clock. In our case, we synchronize the system reset with the system clock.



**Figure 18. Synchronize system reset with system clock**

It is clear to see that when there is the “analog reset”, two flipflops are reset and the output value “system reset” = 0. When we release the analog reset signal, after 2 clock cycles value ‘1’ (VDD) will be sent out to the “system reset”, and it is synchronized with the system clock.

As we indicated in section 2.2, we must combine the system clock (sys\_clk) and the interface clock (i2c\_clk) to generate clock for register file (reg\_clk). When we connect OTP controller to the register file, we use system clock for both OTP controller and register file. When we connect the interface to the register file, we use interface clock for both i2c interface and register file. We use bit “i2c\_busy” to switch between two clock sources. Also, we use a clock gate to turn off system clock after disconnecting OTP controller to the register file. The schematic implement reg\_clk is shown in the following figure.



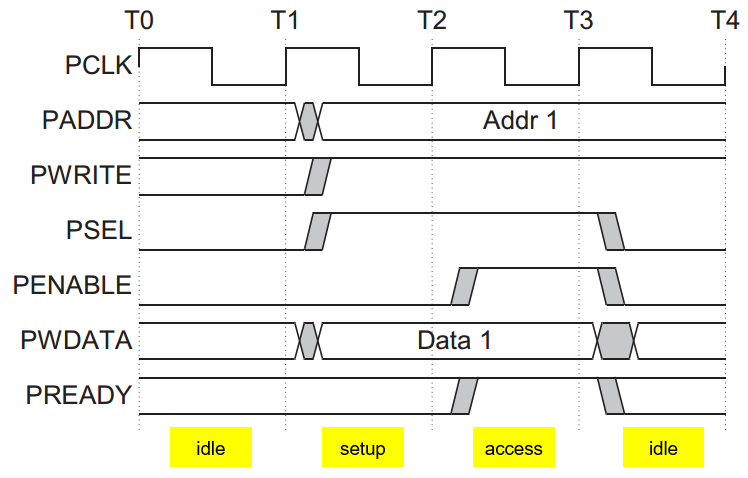
**Figure 19. Combination between system clock and interface clock to register clock**

In the schematic, we use 2 synchronize blocks to sync the enable signal to 2 clock domain sys\_clk and i2c\_clk. The combination logic of otp\_busy, passcode\_en and i2c\_busy is used to generate enable signal for system clock. For enable signal for i2c clock, we use i2c\_busy as a reset signal for synchronize blocks.

It is important to note that all logic gates corresponding to clock, such as clock gating, clock mux, clock or gate, are instantiated from standard cell library with “do\_not\_touch” property. This means the tool does not re-synthesize the logic gates from rtl code. The standard cells are recommened to use medium size.

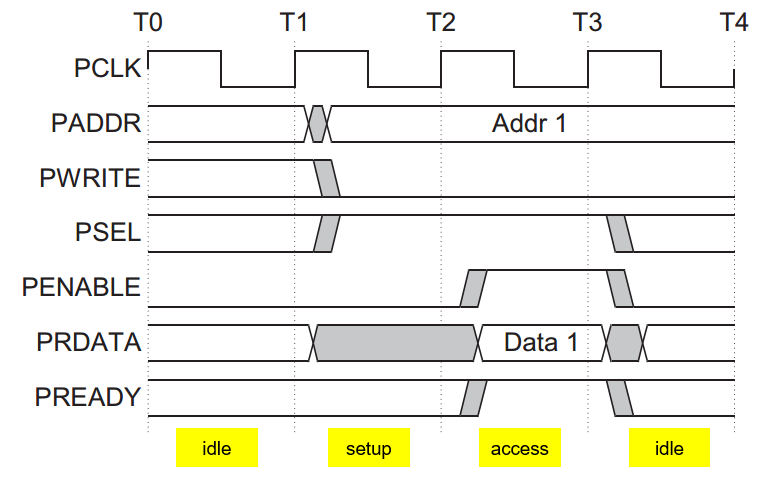
##### APB protocol

APB bus is used to connect low-speed peripheral devices. In usb chip, to perform communication between the register file and i2c controller/otp controller, we design an APB bus between them. I2c controller and otp controller are designed as APB master, while the register file is APB slave. There are 2 main operation in the APB operation: write and read transfer.



**Figure 20. Write transfer**

The write transfer starts with the address (m\_paddr), write data (m\_pwdata) and select signal (m\_psel) all changing after the rising edge of the clock (reg\_clk). The first clock cycle of the transfer is called the Setup phase. After the following clock edge the enable signal is asserted, m\_penable, and this indicates that the Access phase is taking place. The transfer completes at the end of this cycle. At the end of the transfer, the enable signal m\_penable is deasserted.



**Figure 21. Read transfer**

The read transfer starts with the address (m\_paddr), write signal (m\_pwrite) and select signal (m\_psel) all changing after the rising edge of the clock (reg\_clk). Same as write transfer, the first clock cycle of the read transfer is called the Setup phase. After the following clock edge the enable signal is asserted, m\_penable, and this indicates that the Access phase is taking place. The slave must provide data before the end of the read transfer.

**APB master (OTP controller / I2C interface)**

As we presented, there are 4 states in FSM for APB master: m\_idle, m\_setup, m\_access\_write and m\_access\_read. The FSM for APB master is shown in the following figure.



**Figure 22. FSM for APB master**

As we can see in the above figure, the default state of the APB master is always “m\_idle” state. When APB master receives a read or write transfer requirement, we move to “m\_setup” state. In this state, APB master releases the appropriate select signal (m\_psel) and corresponding m\_pwrite (0 for read, 1 for write). We only stay in the “m\_setup” state for 1 clock cycle. If a read transfer is required (read\_n = 0), we move to “m\_access\_read” state. If a write transfer is required (fwrite = 1), we move to “m\_access\_write” state. In “m\_access\_read” state, APB master releases enable signal (m\_penable) and receives data from APB slave via s\_prdata. In “m\_access\_write” state, APB master releases enable signal (m\_penable) and sends data to APB slave via m\_pwdata. Exit from the “m\_access\_\*” state is based on the s\_pready signal from the APB slave. If s\_pready is still low, we stay in “m\_access\_\*” state. If s\_pready is high and there is no transfer requirement, we come back to “m\_ilde” state. If s\_pready is high and there is another transfer requirement, we move to “m\_setup” state.

**APB slave (register file)**

As analysis in APB master, we also have a FSM for APB slave (in the following figure).



**Figure 23. FSM for APB slave**

As we can see in the above figure, there are 4 states in the FSM for APB slave. The default state of the APB slave FSM is “s\_idle” state. If APB slave receives a select signal from APB masterm we move to “s\_setup” state. Besides, in this state, APB slave also receives m\_pwrite signal master and determines next state. If m\_pwrite is high, there is a write transfer, we move to “s\_access\_write” state. In constract, if m\_pwrite is low, there is a read transfer, we move to “s\_access\_read” state. In these states, APB slave reads/writes data from/to corresponding registers and sends back to APB master. After that, if slave is not ready to complete the transaction, it will request the master for some time by deasserting s\_pready. In the situation, we stay in “s\_access\_\*” state. If slave complete the transaction, it assert s\_pready and send to master.



**Figure 24. Connection between APB master and slave**

##### Register distribution

In usb chip, there are 4 types of register: normal register, test register, passcode register and dummy register.

* Normal registers are used in normal mode. It means that in the normal mode, customers can read/write values to the normal registers to control the analog circuits that we allow.
* Test registers are only used in test mode. In the test mode, we do not expect that customers access to the test registers as this can change chip operation (in the worst cse, it can damage the chip). As we discussed, to enter to test mode, we must write a passcode to passcode register. If we are not in the test mode, writing to test resgisters does not take effect.
* Passcode register is only a normal register, but contains security information to enter test mode (refer 2.2.1.4).
* Dummy registers is used for otp programming or reading transaction. In reality, we do not use all fuse elements. Since otp program or read transaction always performs continuously from fist bit (bit 0) to the last bit (bit n-1), we need a dummy register (all values are 0) to “store” or “load”data from/to unused fuse elements.



**Figure 25. OTP distribution in usb chip**

##### Program Inactive mode

In the case of having more than two macros, we need to add program active mode in the FSM. These macros are connected in the same VDDQ power domain. Therefore, while one macro is in program mode, the other macros must be in program inactive mode. To enter program inactive mode, the otp\_csb must be set high when otp\_vddq is at high voltage.

##### Timing parameters

#### DFT scan chain

To implement DFT for usb chip, we need design as following flow:

* Complete normal rtl code for all block designs
* Design test\_en excluded from scan chain
* Design a wrapper for critical signals, such as reset, outputs to analog circuits
* Design clock mux for scan clock, combine to all internal clocks so that all clocks must be changed to scan clock in scan mode
* Design IO controller for pads (combine normal input/output to scan input/output)

##### test\_en

Test en FF is a special flipflop. The FF must be keep 1 in the scan mode (only system reset can change the chip back to normal mode). Therefore it is not in the scan chain (not use scan reset and scan clock). The block diagram of the test en FF is shown in the following figure.



**Figure 26. Schematic of test en generator**

In the boot phase, the system resets test\_en to 0. To switch to scan mode, bit test\_en\_reg is changed to 1. After finishing i2c transaction (i2c\_busy = 0), test\_en is switched to 1 and usb chip moves to scan mode. Because of OR gate appearance, the test\_en signal is keep 1 in the scan mode.

##### Wrapper for critical signals

In the scan mode, some critical signals should be unchanged to avoid taking affect in analog circuits. The signals need to be taked care are reset, clock and outputs of register file.



**Figure 27. Schematic of soft reset generator**

In the I2C interface, there are 4 clocks namely i2c\_clk, i2c\_clk\_n, i2c\_sda\_clk and i2c\_sda\_clk\_n. We will sample data at the rising edge of i2c\_clk clock and send data out at the rising edge of i2c\_clk\_n. Clock i2c\_sda\_clk and i2c\_sda\_clk\_n are used for flipflops of “start\_bit”, “stop\_bit” and “active\_bit”. The schematic of i2c\_clk, i2c\_clk\_n, i2c\_sda\_clk and i2c\_sda\_clk\_n with scan mode is shown in the following figure.



**Figure 28. Schematic of i2c\_clk generator**

For system clock, we also use clock mux to switch system clock and scan clock. The schematic of system clock with scan chain is shown in the following figure.



**Figure 29. Schematic of system clock generator**

When operating in scan mode, the outputs connect directly to analog circuits should be unchanged. These values should be changed to default value. Therefore, we use mux to choose either default value or value get from register file.



**Figure 30. Output to analog circuits generator schematic**

##### IO Controller

There are 4 ports used for scan mode: scan\_in, scan\_out, scan\_en and scan\_clk. To minimize number of ports used in usb chip, we implement the combination between a normal port and a scan port. For example, the following figure presents the schematic used to combine sda port and scan\_out port.



**Figure 31. Implement sda/scan\_out IO Pad**

#### I2C interface

##### I2C Delay element

##### I2C protocol

##### I2C interface reset condition

##### I2C FSM

##### Invert I2C address bit

##### I2C interface schematic

#### Hif\_idle signal

#### Register bus

#### Register MUX

## Requirement

How fast it shall be? How much power it can take? Any area expectation? et cetera