Scripting vs system programming languages

designed to handle different things

- system prog language
  design to manipulate complex
  data structures and algorithms

  e.g. compute ~~circle~~ eigenvectors
  of a matrix

  compute a FFT

- scripting languages
  design to manipulate data

  e.g. read file data
  read I/O device

system-program-lang usually compiled

scripting lang " " interpreted

e.g. sys prog lang : C++, JAVA

scripting lang : python, perl, ruby, tcl

tool command Language (TCL)

pronounced "tickle"

TK is a toolkit for creating graphical I/Fs

Lecture Notes: Tcl Scripting for FPGA Synthesis

DIGITAL ELECTRONICS & INTEGRATED DESIGN AUTOMATION

1. Introduction to Tcl in EDA

Tool Command Language (**Tcl**, pronounced "tickle") is the de facto standard scripting language in the Electronic Design Automation (EDA) industry. Whether you are using AMD Vivado, Intel Quartus Prime, or Synopsys Design Compiler, a native Tcl interpreter is embedded directly into the tool's core executable environment.

Why Use Tcl for Synthesis?

- **Automation:** Eliminates the manual, error-prone process of clicking through complex Graphical User Interfaces (GUIs) for highly repetitive compilation and optimization tasks.
- **Reproducibility:** Guarantees that the identical synthesis constraints, design parameters, and optimization algorithms are applied precisely across different design iterations or multi-developer environments.
- **CI/CD Integration:** Facilitates standard headless execution on remote compute servers, cluster nodes, or within continuous integration frameworks (e.g., Jenkins, GitLab CI).
- **Advanced Data Extraction:** Programmatically parses the internal design database to extract granular timing, utilization, or power metrics that may not be exposed through standard GUI reports.

2. Basic Tcl Syntax Refresher

Before applying Tcl to hardware compilation, it is critical to observe the basic parsing rules of the language:

- **Command Structure:** Every statement is structured as a command followed by space-separated arguments: `command arg1 arg2 ...`
- **Variables:** Declared and updated using the `set` command, and dereferenced/evaluated using the `$` prefix operator.
- **Lists:** Although Tcl treats parameters natively as strings, lists form the baseline structure for handling object groups. They are explicitly declared using the `list` command.
- **Command Substitution:** Square brackets `[...]` nested within a statement evaluate an inner command and pass its return value inline to the outer command.

```
# Variable declaration and evaluation
set project_name "my_filter"
puts "Building project: $project_name"

# Explicit list construction for source tracking
set source_files [list "top.v" "dsp_blk.v" "constraints.xdc"]
```

- TCL is the predominate scripting language in EDA field

- TCL has built in commands for doing File I/O

- TCL has procedures for controling tasks

  e.g. syntheses, PAR, etc

- Xilinx has adopted TCL as the native scripting Lang for ViLvado

for MAC OS, from a command window, type

tclsh (pronounce ticklish)
??

Basic syntax

- cmd <args>
- set a 5 # set a=5
  ↑ not strongly typed; no declarations needed
- set a # return the value of a
- expr 10+20 # return 30
- expr $a+3 # return 8
- set b [expr $a*6]

- `set c {expr $a*6}` #

curly brackets

`set c` # return "expr $a*6"

```
- proc pwr {base p} {
    set result 1
    while {$p >0} {
      set result [expr $result * $base]
      set p [expr $p-1]
    }
      return $result
  }
- pwr 2 6      # return 64
```

other features

- control structures while, for, if
- string manipulation
- I/o on files and I/o devices (e.g. serial port)
- manipulate lisfs
- manipulate arrays
- wait for events to occur

Defn (script)

An ASCII file containing a sequence of TCL commands

(*.tcl extension)

project vs non-project

project mode:

- GUI based (vivado -mode batch -source file.tcl)
- automatically manages the design process
- a project file + directory is created

ON DISK

- source files are copied into the directory
- synthesis & implementation results automatically stored
- only a default set of reports are stored

- non-project mode
  - TCL based
  - you manage the design process
  - compiler type performance
    i.e., in-memory project
    Only what you specify is written to disk
  - you determine what reports are generated

In Non-Project Mode, each design step is controlled using Tcl commands. For example:

- If you modify an HDL file after synthesis, you must remember to rerun synthesis to update the in-memory netlist.
- If you want a timing report after routing, you must explicitly generate the timing repor when routing completes.
- Design parameters and implementation options are set using Tcl commands and parameters.
- You can save design checkpoints and create reports at any stage of the design proces using Tcl.

To put it simply: **Project Mode** is Vivado managing the automation for you, while **Non-Project Mode** is you acting as the automation engine.

Here is a quick breakdown of how this plays out in practice:

1. Project Mode (Automated Scripting)

When you use the Vivado GUI to create a project, add files, and run synthesis/implementation, Vivado is secretly writing TCL commands in the background.

- **The Shortcut:** You don't have to write a script from scratch. Once your project is set up in the GUI, you can simply run `write_project_tcl recreate_project.tcl` in the Tcl Console.
- **The Result:** Vivado spits out a complete, fully formatted TCL script that can recreate your entire project structure, source file links, and run strategies on any machine.

2. Non-Project Mode (Manual Scripting)

In Non-Project Mode, there is no `.xpr` project file keeping track of your source files, IP, or run states. Everything lives purely in the computer's volatile memory during execution.

- **The Process:** Because there is no GUI project to extract a script from, you must write the TCL script yourself using a text editor (like VS Code, Notepad++, or Vim).
- **The Structure:** Your manual script has to explicitly command Vivado through every single step of the hardware compilation compilation pipeline.

Project Mode includes GUI operations, which result in a Tcl command being executed in most cases. The Tcl commands appear in the Vivado IDE Tcl Console and are also captured in the `vivado.jou` file. You can use this file to develop scripts for use with either mode.

Figure 2-3 shows the difference between Project Mode and Non-Project Mode Tcl commands.

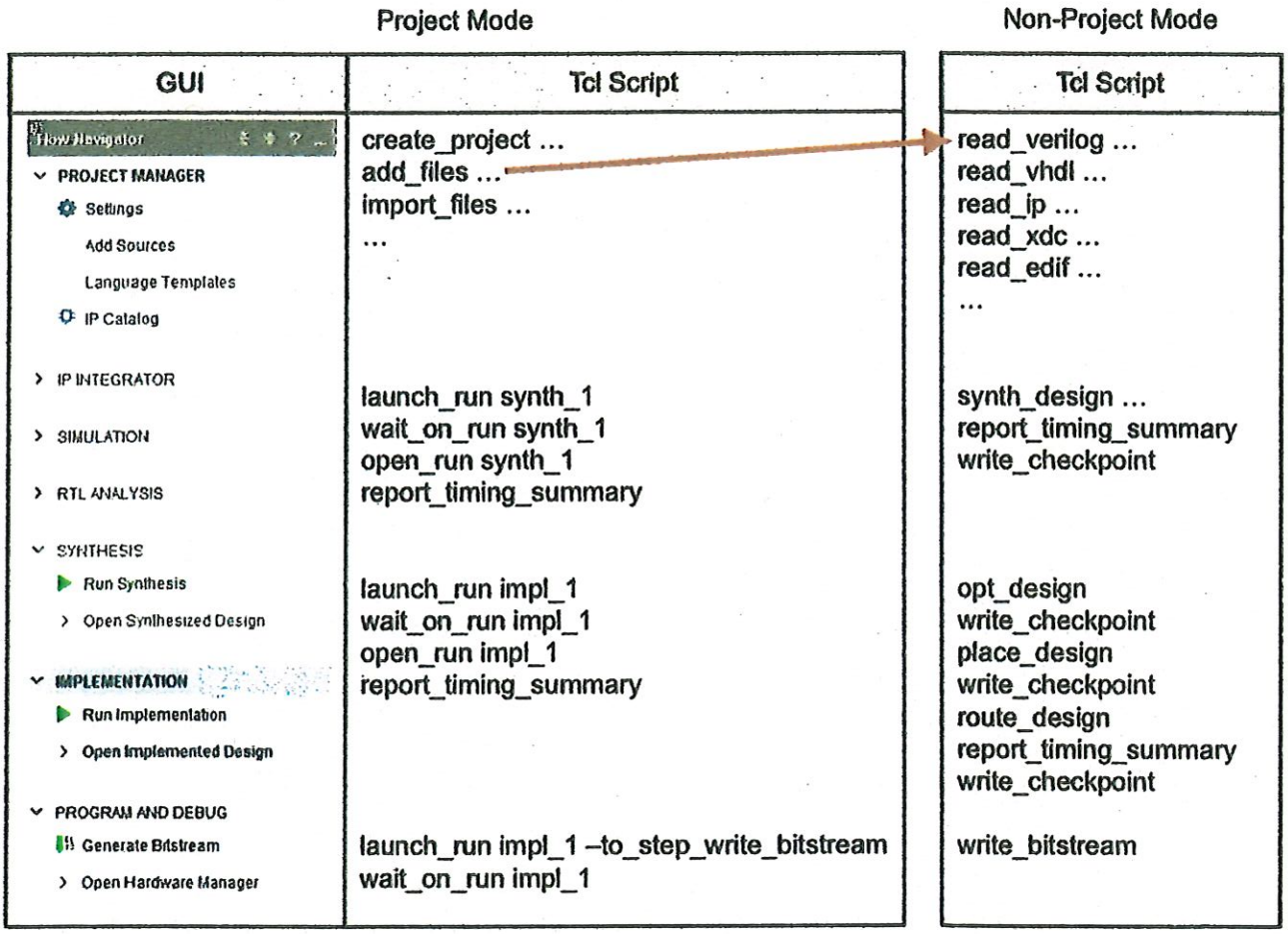


Figure 2-3: **Project Mode and Non-Project Mode Commands**

You add_files in project mode because files are copied into the directory. The non-project mode builds the design in memory so there are no files copied. Hence, you have different Tcl commands.

Yes, from a pure hardware and performance standpoint, you have isolated the absolute core mechanical difference. **Project Mode is disk-heavy; Non-Project Mode is RAM-heavy.** However, because the data is kept in RAM rather than a structured disk database, this mechanical difference completely changes how the software behaves. The choice impacts three major operational areas:

1. The Burden of Automation vs. Manual Direction

Because Project Mode uses the disk to maintain a permanent database file (`.xpr` or `.qpf`), the tool acts as an active manager. It tracks dependencies for you.

- If you modify a Verilog source file on your disk, the tool immediately flags the synthesis step as "out-of-date."
- When you run a wrapper macro like `launch_runs impl_1` , the tool reads the disk, sees synthesis is old, runs synthesis first, saves the result to disk, and then proceeds to placement.

In Non-Project Mode, because everything is just raw data structures floating in volatile RAM, **the tool is completely blind to history.** It doesn't know what you did five minutes ago. If you change a source file, you must manually issue a command to re-read that file into RAM and manually re-trigger `synth_design` , `opt_design` , and `place_design` in your script. The software provides zero automated workflow orchestration.

TCL "glob" command

glob returns a list of thing that match some pattern

e.g.

```
% glob ./*.pdf
```

return a list of all PDFs in the current directory

- **Step 6:** Writes a bitstream to test and program the design onto the Xilinx FPGA.

Compilation with a Project Flow

The following script illustrates a Project flow that synthesizes the design and performs a complete implementation, including bitstream generation. It is based on the CPU example design provided in the Vivado installation tree.

```
#
# STEP#1: define the output directory area.
#
set outputDir ./Tutorial_Created_Data/cpu_project
file mkdir $outputDir
create_project project_cpu_project ./Tutorial_Created_Data/cpu_project \
    -part xc7k70tfbg676-2 -force
#
# STEP#2: setup design sources and constraints
#
add_files -fileset sim_1 ./Sources/hdl/cpu_tb.v
add_files [ glob ./Sources/hdl/bftLib/*.vhdl ]
add_files ./Sources/hdl/bft.vhdl
add_files [ glob ./Sources/hdl/*.v ]
add_files [ glob ./Sources/hdl/mgt/*.v ]
add_files [ glob ./Sources/hdl/or1200/*.v ]
add_files [ glob ./Sources/hdl/usbf/*.v ]
add_files [ glob ./Sources/hdl/wb_conmax/*.v ]
add_files -fileset constrs_1 ./Sources/top_full.xdc
set_property library bftLib [ get_files [ glob ./Sources/hdl/bftLib/*.vhdl ]]
#
# Physically import the files under project_cpu.srcs/sources_1/imports directory
import_files -force -norecurse
#
# Physically import bft_full.xdc under project_cpu.srcs/constrs_1/imports directory
import_files -fileset constrs_1 -force -norecurse ./Sources/top_full.xdc
# Update compile order for the fileset 'sources_1'
set_property top top [current_fileset]
update_compile_order -fileset sources_1
update_compile_order -fileset sim_1
#
# STEP#3: run synthesis and the default utilization report.
#
launch_runs synth_1
wait_on_run synth_1
#
# STEP#4: run logic optimization, placement, physical logic optimization, route and
#         bitstream generation. Generates design checkpoints, utilization and timing
#         reports, plus custom reports.
set_property STEPS.PHYS_OPT_DESIGN.IS_ENABLED true [get_runs impl_1]
set_property STEPS.OPT_DESIGN.TCL.PRE [pwd]/pre_opt_design.tcl [get_runs impl_1]
set_property STEPS.OPT_DESIGN.TCL.POST [pwd]/post_opt_design.tcl [get_runs impl_1]
set_property STEPS.PLACE_DESIGN.TCL.POST [pwd]/post_place_design.tcl [get_runs impl_1]
set_property STEPS.PHYS_OPT_DESIGN.TCL.POST [pwd]/post_phys_opt_design.tcl [get_runs impl_1]
set_property STEPS.ROUTE_DESIGN.TCL.POST [pwd]/post_route_design.tcl [get_runs impl_1]
launch_runs impl_1 -to_step write_bitstream
wait_on_run impl_1
puts "Implementation done!"
```

Compilation and Reporting Example Scripts

Compilation with a Non-Project Flow

The following is an example Tcl script that defines a Non-Project design flow.

The example script uses a custom command `reportCriticalPaths`. This is an illustration on how the Vivado Design Suite can be augmented with custom commands and procedures. The content of `reportCriticalPaths` is provided and explained in the section Defining Tcl Procedures.

```
# STEP#1: define the output directory area.
#
set outputDir ./Tutorial_Created_Data/cpu_output
file mkdir $outputDir
#
# STEP#2: setup design sources and constraints
#
read_vhdl -library bftLib [ glob ./Sources/hdl/bftLib/*.vhdl ]
read_vhdl ./Sources/hdl/bft.vhdl
read_verilog  [ glob ./Sources/hdl/*.v ]
read_verilog  [ glob ./Sources/hdl/mgt/*.v ]
read_verilog  [ glob ./Sources/hdl/or1200/*.v ]
read_verilog  [ glob ./Sources/hdl/usbf/*.v ]
read_verilog  [ glob ./Sources/hdl/wb_conmax/*.v ]
read_xdc ./Sources/top_full.xdc
#
# STEP#3: run synthesis, write design checkpoint, report timing,
# and utilization estimates
#
synth_design -top top -part xc7k70tfbg676-2
write_checkpoint -force $outputDir/post_synth.dcp
report_timing_summary -file $outputDir/post_synth_timing_summary.rpt
report_utilization -file $outputDir/post_synth_util.rpt
#
# Run custom script to report critical timing paths
reportCriticalPaths $outputDir/post_synth_critpath_report.csv
#
# STEP#4: run logic optimization, placement and physical logic optimization,
# write design checkpoint, report utilization and timing estimates
#
opt_design
reportCriticalPaths $outputDir/post_opt_critpath_report.csv
place_design
report_clock_utilization -file $outputDir/clock_util.rpt
#
# Optionally run optimization if there are timing violations after placement
if {[get_property SLACK [get_timing_paths -max_paths 1 -nworst 1 -setup]] < 0} {
    puts "Found setup timing violations => running physical optimization"
    phys_opt_design
}
write_checkpoint -force $outputDir/post_place.dcp
report_utilization -file $outputDir/post_place_util.rpt
report_timing_summary -file $outputDir/post_place_timing_summary.rpt
```

```
#
# NOTE:  typical usage would be "vivado -mode tcl -source run_bft_kintex7_batch.tcl"
#
# STEP#0: define output directory area.
#
set outputDir ./Tutorial_Created_Data/bft_output
file mkdir $outputDir
#
# STEP#1: setup design sources and constraints
#
read_vhdl -library bftLib [ glob ./Sources/hdl/bftLib/*.vhdl ]
read_vhdl ./Sources/hdl/bft.vhdl
read_verilog  [ glob ./Sources/hdl/*.v ]
read_xdc ./Sources/bft_full_kintex7.xdc
#
# STEP#2: run synthesis, report utilization and timing estimates, write checkpoint
design
#
synth_design -top bft -part xc7k70tfbg484-2
write_checkpoint -force $outputDir/post_synth
report_timing_summary -file $outputDir/post_synth_timing_summary.rpt
report_power -file $outputDir/post_synth_power.rpt
#
# STEP#3: run placement and logic optimzation, report utilization and timing
estimates, write checkpoint design
#
opt_design
place_design
phys_opt_design
write_checkpoint -force $outputDir/post_place
report_timing_summary -file $outputDir/post_place_timing_summary.rpt
#
# STEP#4: run router, report actual utilization and timing, write checkpoint design,
run drc, write verilog and xdc out
#
route_design
write_checkpoint -force $outputDir/post_route
report_timing_summary -file $outputDir/post_route_timing_summary.rpt
report_timing -sort_by group -max_paths 100 -path_type summary -file
$outputDir/post_route_timing.rpt
report_clock_utilization -file $outputDir/clock_util.rpt
report_utilization -file $outputDir/post_route_util.rpt
report_power -file $outputDir/post_route_power.rpt
report_drc -file $outputDir/post_imp_drc.rpt
write_verilog -force $outputDir/bft_impl_netlist.v
write_xdc -no_fixed_only -force $outputDir/bft_impl.xdc
#
# STEP#5: generate a bitstream
#
write_bitstream -force $outputDir/bft.bit
```

name of top module (not top file)

has *.dcp extension but it isn't shown

name of file (w/o .py extension)

* my_file.py

file name

Running Synthesis with Tcl

The Tcl command to run synthesis is `synth_design`. Typically, this command is run with multiple options, for example:

```
synth_design -part xc7k30tfbg484-2 -top my_top
```

In this example, `synth_design` is run with the `-part` option and the `-top` option.

In the Tcl Console, you can set synthesis options and run synthesis using Tcl command options. To retrieve a list of options, type `synth_design -help` in the Tcl Console. The following snippet is an example of the `-help` output: `synth_design -help`.

```
Description:
Synthesize a design using Vivado Synthesis and open that design
Syntax:
synth_design  [-name <arg>] [-part <arg>] [-constrset <arg>] [-top <arg>]
              [-include_dirs <args>] [-generic <args>] [-verilog_define
<args>]
              [-seu_max_util <arg>] [-flatten_hierarchy <arg>]
              [-gated_clock_conversion <arg>] [-directive <arg>] [-rtl]
              [-bufg <arg>] [-no_lc] [-fanout_limit <arg>]
             [-shreg_min_size <arg>] [-mode <arg>] [-fsm_extraction <arg>]
              [-keep_equivalent_registers] [-resource_sharing <arg>]
              [-control_set_opt_threshold <arg>] [-max_bram <arg>]
              [-max_dsp <arg>] [-cascade_dsp] [-quiet] [-verbose]
Returns:
design object
Usage:
  Name                              Description
  ------------------------------------------
  [-name]                           Design name
  [-part]                           Target part
  [-constrset]                      Constraint fileset to use
  [-top]                            Specify the top module name
  [-include_dirs]                   Specify verilog search directories
  [-generic]                        Specify generic parameters.
                                    Syntax: -generic
                                    <name>=<value> -generic <name>=<value> ...
  [-verilog_define]                 Specify verilog defines. Syntax:
                                   -verilog_define <macro_name>[=<macro_text>]
                                   -verilog_define <macro_name>[=<macro_text>]
  [-flatten_hierarchy]              Flatten hierarchy during LUT mapping.
                                    Values:
                                    full, none, rebuilt
                                    Default: rebuilt
  [-gated_clock_conversion]         Convert clock gating logic to flop enable.
                                    Values: off, on, auto
                                    Default: off
  [-directive]                      Synthesis directive. Values:
                                    default,
```

	AreaOptimized_high, AreaMutlThresholdDSP AlternateRoutability FewerCarryChains RunTimeOptimized Default: default
[-rtl]	Elaborate and open an rtl design
[-bufg]	Max number of global clock buffers used by synthesis Default: 12
[-no_lc]	Disable LUT combining. Do not allow combining LUT pairs into single dual output LUTs.
[-fanout_limit]	Fanout limit. This switch does not impact control signals (such as set,reset, clock enable) use MAX_FANOUT to replicate these signals if needed. Default: 10000
[-shreg_min_size]	Minimum length for chain of registers to be mapped onto SRL Default: 3
[-mode]	The design mode. Values: default, out_of_context Default: default
[-fsm_extraction]	FSM Extraction Encoding. Values: off, one_hot, sequential, johnson, gray, auto Default: auto
[-keep_equivalent_registers]	Prevents registers sourced by the same logic from being merged. (Note that the merging can otherwise be prevented using the synthesis KEEP attribute)
[-resource_sharing]	Sharing arithmetic operators. Value: auto, on, off Default: auto
[-control_set_opt_threshold]	Threshold for synchronous control set optimization to lower number of control sets. Valid values are 'auto', integer 0 to 16. The higher the number, the more control set optimization will be performed and fewer control sets will result. To disable control set optimization completely, set to 0. Default: auto
[-max_bram]	Maximum number of block RAM allowed in design. (Note -1 means that the tool will choose the max number allowed for the part in question Default: -1
[-max_dsp]	Maximum number of block DSP allowed in design. (Note -1 means that the tool will choose the max number allowed for the part Default: -1

```
[-cascade_dsp]          Controls how adders in sum DSP block outputs
                        are implemented.
                        The values are: auto, tree, and force.
[-quiet]                Ignore command errors
[-verbose]              Suspend message limits during execution
```

For the `-generic` option, special handling needs to happen with VHDL boolean and `std_logic` vector type because those type do not exist in other formats. Instead of `TRUE`, `FAlse`, or 0010, for example, Verilog standards should be given.

For boolean, the value for `FALSE` is as follows:

```
-generic my_gen=1'b0
```

For `std_logic` vector the value for 0010 is:

```
-generic my_gen=4'b0010
```

IMPORTANT: *Overriding string generics or parameters is not supported.*

IMPORTANT: *If you are using the* `-mode out_of_context` *option on the top-level, do not use the* `PACKAGE_PIN` *property unless there is an I/O buffer instantiated in the RTL. The out_of_context option tells the tool to not infer any I/O buffers including tristate buffers. Without the buffer, you will get errors in placer.*

A verbose version of the help is available in the *Vivado Design Suite: Tcl Command Reference Guide* (UG835) [Ref 3]. To determine any Tcl equivalent to a Vivado IDE action, run the command in the Vivado IDE and review the content in the TCL Console or the log file.

Multithreading in RTL Synthesis

On multiprocessor systems, RTL synthesis leverages multiple CPU cores by default (up to 4) to speed up compile times.

The maximum number of simultaneous threads varies, depending on the number of processors available on the system, the OS, and the stage of the flow (see this link in the Implementation User Guide - UG904). The Tcl parameter `general.maxThreads`, which is common to all threads in Vivado, gives control to the user to specify the number of threads to use when running RTL synthesis. For example:

```
Vivado% set_param general.maxThreads <new limit>
```

Where the `<new limit>` must be an integer from 1 to 8 inclusive. For RTL synthesis, 4 is the maximum number of threads that can be set effectively.

Defn (FPGA image)

a binary file thats the programming data for the FPGA

Note:

image = config. bitstream

= programming bitstream

= bitstream

****

In 2nd video concentrate on daisy chaining start @ 20:49

How-to-configure-an-FPGA Xilinx video

Show summary slide nach

So now the question becomes this:

How do I program those flash memories—especially if they're soldered on a PCB?

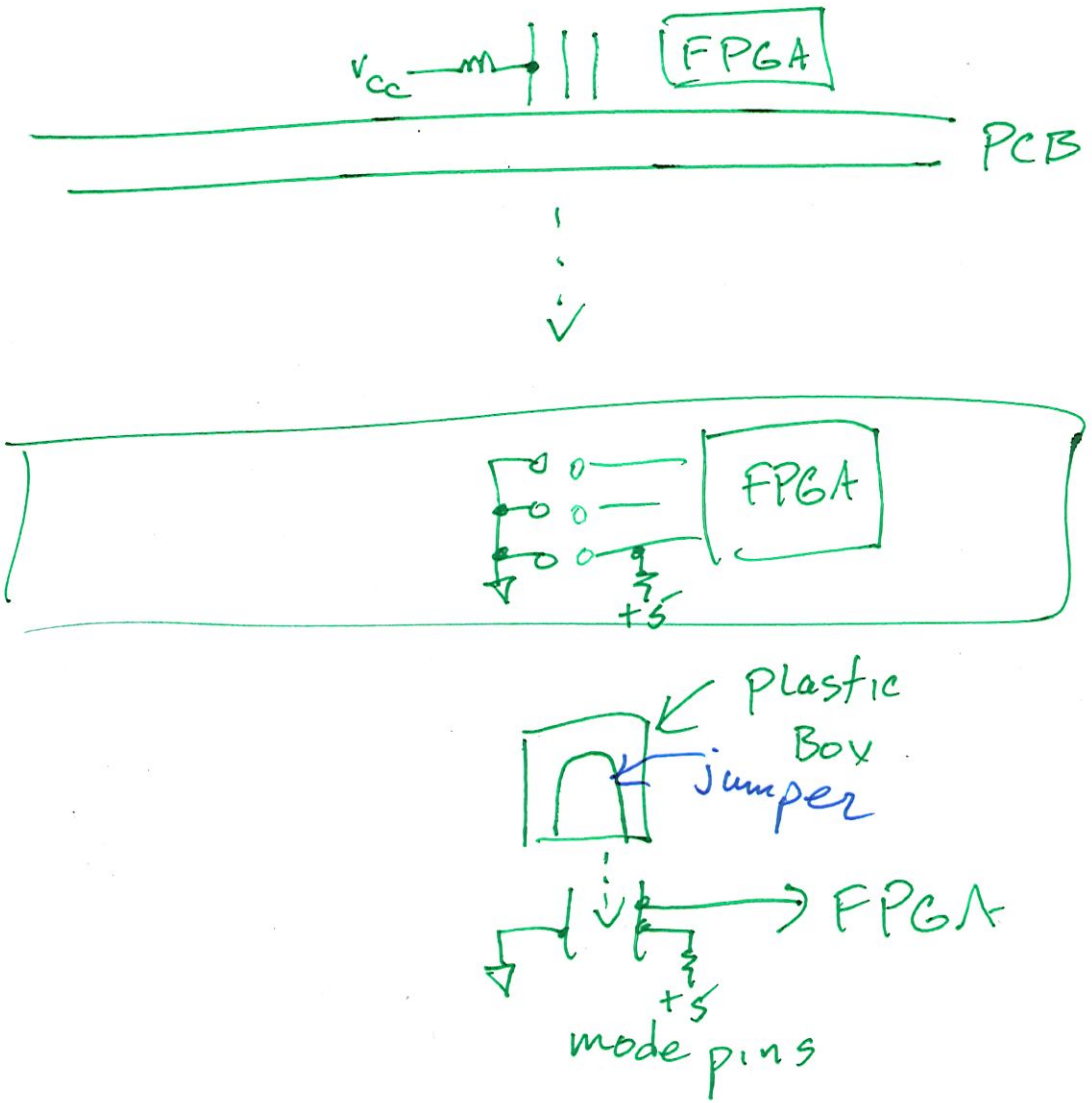
Answer: JTAG

****

indirect programming of BPI PROMs with Virtex-5 application note

7 series ~~Virtex~~ 7

config doc


V_{cc}
FPGA
PCB
FPGA
+5
plastic
Box
jumper
FPGA
+5
mode pins

7 Series FPGAs Configuration

User Guide

UG470 (v1.17) December 5, 2023

AMD Adaptive Computing is creating an environment where employees, customers, and partners feel welcome and included. To that end, we're removing non-inclusive language from our products and related collateral. We've launched an internal initiative to remove language that could exclude people or reinforce historical biases, including terms embedded in our software and IPs. You may still find examples of non-inclusive language in our older products as we work to make these changes and align with evolving industry standards. Follow this link for more information.

AMD

Table of Contents

About This Guide

AMD 7 series FPGAs include four FPGA families that are all designed for lowest power to enable a common design to scale across families for optimal power, performance, and cost. The AMD Spartan™-7 family is the lowest density with the lowest cost entry point into the 7 series portfolio. The AMD Artix™-7 family is optimized for highest performance-per-watt and bandwidth-per-watt for cost-sensitive, high-volume applications. The AMD Kintex™-7 family is an innovative class of FPGAs optimized for the best price-performance. The AMD Virtex™-7 family is optimized for highest system performance and capacity. This guide serves as a technical reference describing the 7 series FPGAs configuration.

This *7 Series FPGAs Configuration User Guide* is part of an overall set of documentation on the 7 series FPGAs, which is available on the AMD website at www.xilinx.com/documentation.

Guide Contents

This manual contains these topics:

AMD

Chapter 1

Configuration Overview

This chapter provides a brief overview of the 7 series FPGA configuration methods and features. Subsequent chapters provide more detailed descriptions of each configuration method and feature. The configuration methods and features described herein are available on all family members with few exceptions.

Overview

AMD 7 series FPGAs are configured by loading application-specific configuration data—a bitstream—into internal memory. 7 series FPGAs can load themselves from an external nonvolatile memory device or they can be configured by an external smart source, such as a microprocessor, DSP processor, microcontroller, PC, or board tester. In any case, there are two general configuration datapaths. The first is the serial datapath that is used to minimize the device pin requirements. The second datapath is the 8-bit, 16-bit, or 32-bit datapath used for higher performance or access (or link) to industry-standard interfaces, ideal for external data sources like processors, or x8- or x16-parallel flash memory.

Like processors and processor peripherals, AMD FPGAs can be reprogrammed, in system, on demand, an unlimited number of times.

Because AMD FPGA configuration data is stored in CMOS configuration latches (CCLs), it must be reconfigured after it is powered down. The bitstream is loaded each time into the device through special configuration pins. These configuration pins serve as the interface for a number of different configuration modes:

- Master-Serial configuration mode
- Slave-Serial configuration mode
- Master SelectMAP (parallel) configuration mode (x8 and x16)
- Slave SelectMAP (parallel) configuration mode (x8, x16, and x32)
- JTAG/boundary-scan configuration mode
- Master Serial Peripheral Interface (SPI) flash configuration mode (x1, x2, x4)
- Master Byte Peripheral Interface (BPI) flash configuration mode (x8 and x16) using parallel NOR flash

The configuration modes are explained in detail in Chapter 2, Configuration Interfaces.

The specific configuration mode is selected by setting the appropriate level on the dedicated mode input pins M[2:0]. The M2, M1, and M0 mode pins should be set at a constant DC voltage level, either through pull-up or pull-down resistors (≤ 1 kΩ), or tied directly to ground or V_{CCO_0}. The mode pins should not be toggled during and after configuration. See Chapter 2, Configuration Interfaces for the mode pin setting options.

The terms Master and Slave refer to the direction of the configuration clock (CCLK):

- In Master configuration modes, the 7 series device drives CCLK from an internal oscillator. To select the desired frequency, the bitstream `-g ConfigRate` option is used. The BitGen section of UG628, *ISE Command Line Tools User Guide* provides more information for the AMD ISE™ Design Suite. The Device Configuration Bitstream Settings section of UG908, *Vivado Programming and Debugging User Guide* provides more information for the AMD Vivado™ Design Suite. After configuration, the CCLK is turned OFF unless the persist option is selected or SEU detection is used. See Persist Option in Chapter 6. The CCLK pin is 3-stated with a weak pull-up.
- In Slave configuration modes, CCLK is an input.

The JTAG/boundary-scan configuration interface is always available, regardless of the mode pin settings.

7 Series FPGAs Configuration Differences from Previous FPGA Generations

The 7 series devices support the same configuration interfaces supported on AMD Virtex™-6 FPGAs except for the Master BPI-Down mode. Master BPI-Down mode is not supported in the 7 series FPGAs. In addition, a few of the configuration interfaces are enhanced with these features that enable faster configuration:

- The Master SPI configuration mode supports reading from an SPI flash using a data bus up to four bits wide, which is similar to the AMD Spartan™-6 FPGA Master SPI configuration mode.

  Note: In the 7 series, the DIN pin function is assigned to a multi-function pin that shares the D01 configuration data bus pin in order to support the x2 or x4 SPI data widths. This is different from the AMD Virtex-6 FPGA where DIN was a dedicated pin, and this is different from the AMD Spartan-6 FPGA where DIN was assigned to the multi-purpose D0 configuration data bus pin.
- The Master SPI configuration mode supports clocking data on the negative edge, allowing for optimal use of the clock period and therefore faster configuration speed.
- The Master SPI configuration mode supports flash densities greater than 128 Mb.
- The Master BPI configuration mode supports reading from a BPI (parallel NOR) flash via the flash device's burst, synchronous read mode. The ADV_B pin is new relative to the AMD Virtex-6 FPGA BPI interface to support the address latching required for the BPI synchronous read mode.
- The AES decryptor supports configuration data bus widths up to 16 bits wide.
- Relative to AMD Virtex-6, the SelectMAP modes and ICAPE2 primitive do not have a BUSY pin/port. BUSY is not needed in the 7 series because the SelectMAP/ICAPE2 output data is deterministic (see Accessing Configuration Registers through the SelectMAP Interface, page 120.) See 7 series data sheet for ICAPE2 maximum frequency (F_{ICAPCK}).
- See UG953, *Vivado Design Suite 7 Series FPGA and Zynq 7000 SoC Libraries Guide* for the configuration and boundary scan components (primitives). The 7 series primitive names end with an "E2" suffix, whereas the AMD Virtex-6 FPGA primitives ended with the "_VIRTEX6" suffix.

The 7 series devices support configuration interfaces with 3.3V, 2.5V, 1.8V, or 1.5V I/O. The configuration interfaces include the JTAG pins in bank 0, the dedicated configuration pins in bank 0, and the pins related to specific configuration modes in bank 14 and bank 15. To support the appropriate configuration interface voltage on bank 0, bank 14, and bank 15, the following is required:

- The configuration banks voltage select pin (CFGBVS) must be set to a High (V_{CCO_0}) or Low (GND) to set the configuration and JTAG I/O in banks 0, 14, and 15 for 3.3V/2.5V or 1.8V/1.5V operation, respectively. When CFGBVS is set to Low for 1.8V/1.5V I/O operation, the V_{CCO_0} supply and I/O signals to bank 0 must be 1.8V (or lower) to avoid device damage. If CFGBVS is Low, then any I/O pins used for configuration in banks 14 and 15 must also be powered and operated at 1.8V or 1.5V. See Configuration Banks Voltage Select, page 28 for further details.

The operating voltage of the I/O in bank 14 and bank 15 are determined by the V_{CCO_14} and V_{CCO_15} supplies, respectively. When bank 14 or bank 15 are used for configuration, the V_{CCO} supplies for the applicable banks should match the V_{CCO_0} voltage for voltage compatibility across the configuration interface. When CFGBVS is tied to GND for 1.8V/1.5V I/O operation, then if any configuration I/O are used in bank 14 or bank 15, VCCO_14 or VCCO_15 and the configuration I/O signals to bank 14 or bank 15 must be 1.8V or 1.5V to avoid device damage.

Most 7 series FPGAs are supported by both the ISE Design Suite, which also supports previous generations, and the newer Vivado Design Suite. The user options described in this user guide generally refer to the ISE Design Suite tool names, but the same options are found in the Vivado Design Suite. For example, the ISE Design Suite BitGen tool generates bitstreams. In Vivado, the WRITE_BITSTREAM Tcl command can be used. For more information, see:

- UG835, *Vivado Design Suite Tcl Command Reference Guide*
- UG908, *Vivado Design Suite User Guide: Programming and Debugging*

Note: The BitGen command options are Tcl properties in the Vivado Design Suite. See Appendix A, *Device Configuration Bitstream Settings,* in UG908 for details on the properties and values.

Design Considerations

To make an efficient system, it is important to consider which FPGA configuration mode best matches the system's requirements. Each configuration mode dedicates certain FPGA pins and can temporarily use other multi-function pins during configuration only. These multi-function pins are then released for general use when configuration is completed. Similarly, the configuration mode can place voltage restrictions on some FPGA I/O banks. Several different configuration options are available, and while the options are flexible, there is often an optimal solution for each system. Several topics must be considered when choosing the best configuration option: overall setup, speed, cost, and complexity.

Configuration Bitstream Lengths

FPGA designs are compiled into bitstreams. The bitstreams are loaded through a configuration interface to configure the FPGA with the design. A complete bitstream for each FPGA part type has a fixed length. Table 1-1 shows the bitstream lengths and other device-specific information for the 7 series FPGAs.

Table 1-1: **Bitstream Length**

Device	Configuration Bitstream Length (bits)	Minimum Configuration Flash Memory Size (Mb)	JTAG/Device IDCODE[31:0] (hex)[1]	Production IDCODE Revision	JTAG Instruction Length (bits)	Super Logic Regions
AMD Spartan 7 Family						
7S6	4,310,752	8	X3622093	0 or later	6	N/A
7S15	4,310,752	8	X3620093	0 or later	6	N/A
7S25	9,934,432	16	X37C4093	0 or later	6	N/A
7S50	17,536,096	32	X362F093	0 or later	6	N/A
7S75	29,494,496	32	X37C8093	0 or later	6	N/A
7S100	29,494,496	32	X37C7093	0 or later	6	N/A
AMD Artix 7 Family						
7A12T	9,934,432	16	X37C3093	0 or later	6	N/A
7A15T	17,536,096	32	X362E093	0 or later	6	N/A
7A25T	9,934,432	16	X37C2093	0 or later	6	N/A
7A35T	17,536,096	32	X362D093	0 or later	6	N/A
7A50T	17,536,096	32	X362C093	0 or later	6	N/A
7A75T	30,606,304	32	X3632093	1 or later	6	N/A
7A100T	30,606,304	32	X3631093	1 or later	6	N/A
7A200T	77,845,216	128	X3636093	1 or later	6	N/A
AMD Kintex 7 Family						
7K70T	24,090,592	32	X3647093	0 or later	6	N/A
7K160T	53,540,576	64	X364C093	0 or later	6	N/A
7K325T	91,548,896	128	X3651093	4 or later	6	N/A
7K355T	112,414,688	128	X3747093	0 or later	6	N/A
7K410T	127,023,328	128	X3656093	1 or later	6	N/A
7K420T	149,880,032	256	X3752093	2 or later	6	N/A
7K480T	149,880,032	256	X3751093	2 or later	6	N/A
AMD Virtex 7 Family						
7V585T	161,398,880	256	X3671093	0 or later	6	N/A
7V2000T	447,337,216	512	X36B3093[2]	2 or later	24	4
7VX330T	111,238,240	128	X3667093	0 or later	6	N/A
7VX415T	137,934,560	256	X3682093	3 or later	6	N/A
7VX485T	162,187,488	256	X3687093	3 or later	6	N/A
7VX550T	229,878,496	256	X3692093	3 or later	6	N/A

Table 1-1: **Bitstream Length *(Cont'd)***

Device	Configuration Bitstream Length (bits)	Minimum Configuration Flash Memory Size (Mb)	JTAG/Device IDCODE[31:0] (hex)(1)	Production IDCODE Revision	JTAG Instruction Length (bits)	Super Logic Regions
7VX690T	229,878,496	256	X3691093	3 or later	6	N/A
7VX980T	282,521,312	512	X3696093	0 or later	6	N/A
7VX1140T	385,127,680	512	X36D5093	2 or later	24	4
7VH580T	195,663,008	256	X36D9093	2 or later	22	2
7VH870T	294,006,336	512	X36DB093	2 or later	38	3

Notes:

1. The 'X' in the JTAG IDCODE value represents the revision field (IDCODE[31:28]) which can vary.
2. The 7V2000T IDCODE contains additional don't care ('X') bits beyond the revision field. The complete binary IDCODE[31:0] value with don't care bit positions is: `XXXX_0011_0110_1011_XX11_0000_1001_0011`.

FPGA Configuration Data Source

AMD 7 series FPGAs are designed for maximum flexibility. The FPGA either automatically loads itself with configuration data from a nonvolatile flash memory, or another external intelligent device such as a processor or microcontroller can download the configuration data to the FPGA. In addition, the configuration data can be downloaded from a host computer through a cable to the JTAG port of the FPGA.

Master Modes

The self-loading FPGA configuration modes, generically called *Master* modes, are available with either a serial or parallel datapath. The Master modes leverage various types of nonvolatile memories to store the FPGA's configuration information. In Master mode, the FPGA's configuration bitstream typically resides in nonvolatile memory on the same board, generally external to the FPGA. The FPGA generates a configuration clock signal in an internal oscillator that drives the configuration logic and is visible on the CCLK output pin. The FPGA controls the configuration process.

Slave Modes

The externally controlled loading FPGA configuration modes, generically called *Slave* modes, are also available with either a serial or parallel datapath. In Slave mode, an external "intelligent agent" such as a processor, microcontroller, DSP processor, or tester downloads the configuration image into the FPGA, as shown in Figure 1-1. The advantage of the Slave configuration modes is that the FPGA bitstream can reside almost anywhere in the overall system. The bitstream can reside in flash, onboard, along with the host processor's code. It can reside on a hard disk. It can originate somewhere over a network connection or another type of bridge connection.

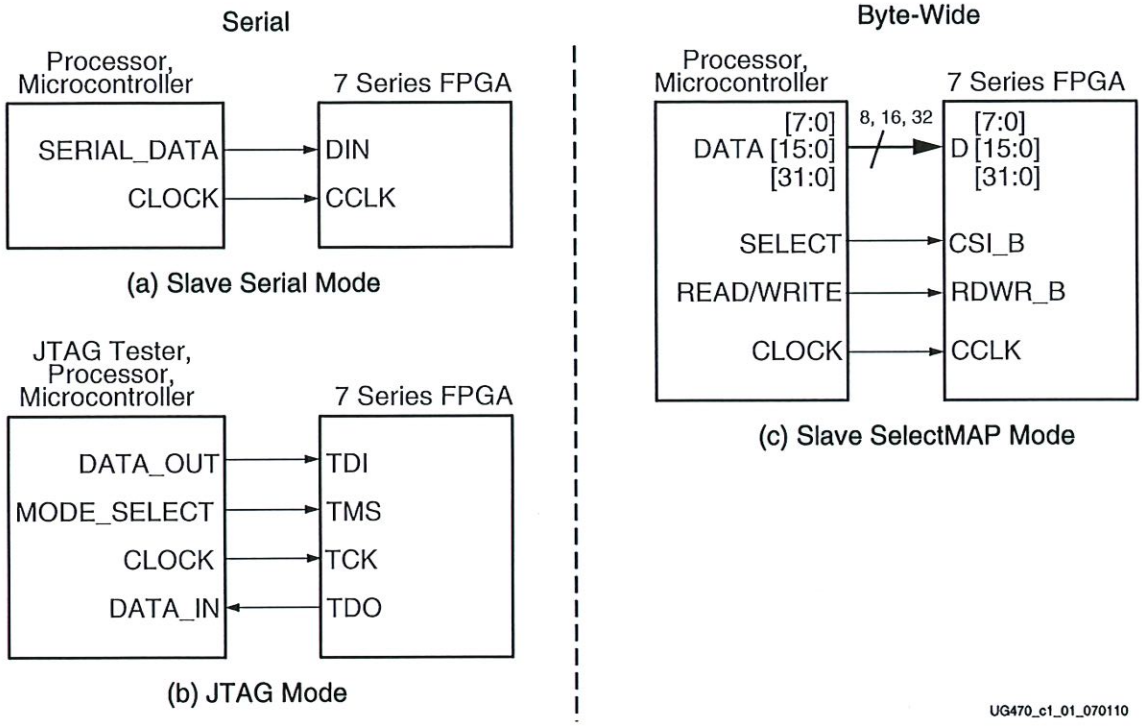


Figure 1-1: **Slave Configuration Modes**

The Slave Serial mode is extremely simple, consisting only of a clock and serial data input. The JTAG mode is also a simple serial configuration mode, popular for prototyping and highly utilized for board test. The Slave SelectMAP mode is a simple x8-, x16-, or x32-bit-wide processor peripheral interface, including a chip-select input and a read/write control input.

JTAG Connection

The four-pin JTAG interface is common on board testers and debugging hardware. In fact, the AMD programming cable for 7 series FPGAs, listed here, uses the JTAG interface for prototype download and debugging. Regardless of the configuration mode ultimately used in the application, it is best to also include a JTAG configuration path for easy design development. Also see Chapter 3, Boundary-Scan and JTAG Configuration.

- **Platform Cable USB II**
  https://www.xilinx.com/products/boards-and-kits/hw-usb-ii-g.html

The Basic Configuration Solution

In the basic configuration solution, the FPGA automatically retrieves its bitstream from a flash memory device at power-on. The FPGA has a serial peripheral interface (SPI) through which the FPGA can read a bitstream from a standard SPI flash device.

The iMPACT tool in the ISE software can program select SPI flash memories. The iMPACT tool can communicate with the FPGA through its standard JTAG interface and can program the SPI flash, indirectly through the FPGA. See XAPP586, *Using SPI Flash with 7 Series FPGAs*. Similar capability is found in the Vivado device programmer.

AMD

Chapter 2

Configuration Interfaces

AMD 7 series devices have five configuration interfaces. Each configuration interface corresponds to one or more configuration modes and bus width, shown in Table 2-1. For detailed interface timing information, see the respective 7 series FPGAs data sheet. Configuration timing is relative to the CCLK at the pin, even in Master modes where the CCLK is generated internally.

Table 2-1: **7 Series FPGA Configuration Modes**

Configuration Mode	M[2:0]	Bus Width	CCLK Direction
Master Serial	000	x1	Output
Master SPI	001	x1, x2, x4	Output
Master BPI	010	x8, x16	Output
Master SelectMAP	100	x8, x16	Output
JTAG	101	x1	Not Applicable
Slave SelectMAP	110	x8, x16, x32(1)	Input
Slave Serial(2)	111	x1	Input

Notes:

1. The Slave SelectMAP x16 and x32 bus widths do not support AES-encrypted bitstreams.
2. This is the default setting due to internal pull-up resistors on the Mode pins.

Configuration Pins

Each configuration mode has a corresponding set of interface pins that span one or more I/O banks on the 7 series FPGA. Bank 0 contains the dedicated configuration pins and is always part of every configuration interface. Bank 14 and Bank 15 contain multi-function pins that are involved in specific configuration modes. The 7 series FPGAs data sheets specify the switching characteristics for configuration pins in banks operating at 3.3V, 2.5V, 1.8V, or 1.5V.

All JTAG and dedicated configuration pins are located in a separate, dedicated bank with a dedicated voltage supply (VCCO_0). The multi-function pins are located in Banks 14 and 15.

All dedicated input pins operate at the VCCO_0 LVCMOS level (LVCMOS15, LVCMOS18, LVCMOS25, or LVCMOS33). All active dedicated output pins operate at the VCCO_0 voltage level with the output standard set to LVCMOS, 12 mA drive, fast slew rate. For all modes that use multi-function I/O, the associated VCCO_14 or VCCO_15 must be connected to the appropriate voltage to match the I/O standard of the configuration device. The multi-function pins are also LVCMOS, 12 mA drive, fast slew rate during configuration. If the Persist option is used (See Persist Option, page 120), the multi-function I/O for the selected configuration mode remain active after

By hardwiring $M[2:0] = 101$, you explicitly instruct the FPGA: **"Do not attempt to read from any external interface at power-up. Do not drive any configuration clocks. Sit quietly and wait indefinitely for a JTAG command."**

configuration, with the I/O standard set to the general-purpose default of LVCMOS, 12 mA drive, Slow slew rate.

Table 2-2 and Table 2-3 show the configuration mode pins and their location across the I/O banks.

Table 2-2: **Configuration Mode Pins (Table 1 of 2)**

Pin Name	Bank	JTAG (Only)	Slave Serial	Master Serial	Master SPI		
					x1	x2	x4
CFGBVS	0	CFGBVS	CFGBVS	CFGBVS	CFGBVS	CFGBVS	CFGBVS
M[2:0]	0	M[2:0]=101	M[2:0]=111	M[2:0]=000	M[2:0]=001	M[2:0]=001	M[2:0]=001
TCK	0	TCK	TCK	TCK	TCK	TCK	TCK
TMS	0	TMS	TMS	TMS	TMS	TMS	TMS
TDI	0	TDI	TDI	TDI	TDI	TDI	TDI
TDO	0	TDO	TDO	TDO	TDO	TDO	TDO
PROGRAM_B	0	PROGRAM_B	PROGRAM_B	PROGRAM_B	PROGRAM_B	PROGRAM_B	PROGRAM_B
INIT_B	0	INIT_B	INIT_B	INIT_B	INIT_B	INIT_B	INIT_B
DONE	0	DONE	DONE	DONE	DONE	DONE	DONE
CCLK	0	CCLK	CCLK	CCLK	CCLK	CCLK	CCLK
PUDC_B(1)	14	PUDC_B(1)	PUDC_B(1)	PUDC_B(1)	PUDC_B(1)	PUDC_B(1)	PUDC_B(1)
EMCCLK(2)	14	–	–	EMCCLK(2)	EMCCLK(2)	EMCCLK(2)	EMCCLK(2)
CSI_B	14	–	–	–	–	–	–
DOUT_CSO_B(3)(4)	14	–	[DOUT](3)	[DOUT](3)	[DOUT](3)	–	–
RDWR_B	14	–	–	–	–	–	–
FCS_B	14	–	–	–	FCS_B	FCS_B	FCS_B
D00_MOSI	14	–	–	–	MOSI	MOSI/D00	MOSI/D00
D01_DIN	14	–	DIN	DIN	DIN	DIN/D01	DIN/D01
D02	14	–	–	–	–	–	D02
D03	14	–	–	–	–	–	D03
D[04-07]	14	–	–	–	–	–	–
D[08-15]	14	–	–	–	–	–	–
A[00-15]_D[16-31]	14	–	–	–	–	–	–
A[16-28]	15	–	–	–	–	–	–
FOE_B	15	–	–	–	–	–	–
FWE_B	15	–	–	–	–	–	–
ADV_B	15	–	–	–	–	–	–
RS0, RS1(5)	15	RS0, RS1(5)	RS0, RS1(5)	RS0, RS1(5)	–	–	–

Notes:

1. PUDC_B has special functionality during configuration but is independent of all configuration interfaces, i.e. PUDC_B does not need to be voltage compatible with other pins in a configuration interface.
2. EMCCLK is only used when the ExtMasterCclk_en option enables EMCCLK as an input for clocking the master configuration modes.
3. DOUT is only used in a serial configuration daisy-chain for outputting data to the downstream FPGA (or for the DebugBitstream option). Otherwise, DOUT is high-Z.
4. CSO_B is only used in a parallel configuration daisy-chain for outputting a chip-enable signal to a downstream device. Otherwise, CSO_B is high-Z.
5. RS0 and RS1 are only driven when a MultiBoot event is initiated or when the ConfigFallback option is enabled and a Fallback event occurs. Otherwise, RS0 and RS1 are high-Z. When using the RS[1:0] pins for configuration it is recommended not to use them in User mode.
6. Empty cells indicate that the pin is not used in the configuration mode and is ignored and is high-Z during configuration.

Table 2-3: **Configuration Mode Pins (Table 2 of 2)**

Pin Name	Bank	Master SelectMAP		Slave SelectMAP			Master BPI	
		x8	x16	x8	x16	x32	x8	x16
CFGBVS	0	CFGBVS	CFGBVS	CFGBVS	CFGBVS	CFGBVS	CFGBVS	CFGBVS
M[2:0]	0	M[2:0]=100	M[2:0]=100	M[2:0]=110	M[2:0]=110	M[2:0]=110	M[2:0]=010	M[2:0]=010
TCK	0	TCK	TCK	TCK	TCK	TCK	TCK	TCK
TMS	0	TMS	TMS	TMS	TMS	TMS	TMS	TMS
TDI	0	TDI	TDI	TDI	TDI	TDI	TDI	TDI
TDO	0	TDO	TDO	TDO	TDO	TDO	TDO	TDO
PROGRAM_B	0	PROGRAM_B	PROGRAM_B	PROGRAM_B	PROGRAM_B	PROGRAM_B	PROGRAM_B	PROGRAM_B
INIT_B	0	INIT_B	INIT_B	INIT_B	INIT_B	INIT_B	INIT_B	INIT_B
DONE	0	DONE	DONE	DONE	DONE	DONE	DONE	DONE
CCLK	0	CCLK	CCLK	CCLK	CCLK	CCLK	CCLK	CCLK
PUDC_B(1)	14	PUDC_B(1)	PUDC_B(1)	PUDC_B(1)	PUDC_B(1)	PUDC_B(1)	PUDC_B(1)	PUDC_B(1)
EMCCLK(2)	14	EMCCLK(2)	EMCCLK(2)	–	–	–	EMCCLK(2)	EMCCLK(2)
CSI_B	14	CSI_B	CSI_B	CSI_B	CSI_B	CSI_B	–	–
DOUT_CSO_B(3)(4)	14	[CSO_B](4)	[CSO_B](4)	[CSO_B](4)	[CSO_B](4)	[CSO_B](4)	[CSO_B](4)	[CSO_B](4)
RDWR_B	14	RDWR_B	RDWR_B	RDWR_B	RDWR_B	RDWR_B	–	–
FCS_B	14	–	–	–	–	–	FCS_B	FCS_B
D00_MOSI	14	D00	D00	D00	D00	D00	D00	D00
D01_DIN	14	D01	D01	D01	D01	D01	D01	D01
D02	14	D02	D02	D02	D02	D02	D02	D02
D03	14	D03	D03	D03	D03	D03	D03	D03
D[04-07]	14	D[04-07]	D[04-07]	D[04-07]	D[04-07]	D[04-07]	D[04-07]	D[04-07]
D[08-15]	14	–	D[08-15]	–	D[08-15]	D[08-15]	–	D[08-15]
A[00-15]_D[16-31]	14	–	–	–	–	D[16-31]	A[00-15]	A[00-15]
A[16-28]	15	–	–	–	–	–	A[16-28]	A[16-28]
FOE_B	15	–	–	–	–	–	FOE_B	FOE_B
FWE_B	15	–	–	–	–	–	FWE_B	FWE_B
ADV_B	15	–	–	–	–	–	ADV_B	ADV_B
RS0, RS1(5)	15	RS0, RS1(5)	RS0, RS1(5)	RS0, RS1(5)	RS0, RS1(5)	RS0, RS1(5)	RS0, RS1(5)	RS0, RS1(5)

Notes:

1. PUDC_B has special functionality during configuration but is independent of all configuration interfaces, i.e. PUDC_B does not need to be voltage compatible with other pins in a configuration interface.
2. EMCCLK is only used when the BitGen ExtMasterCclk_en option enables EMCCLK as an input for clocking the master configuration modes.
3. DOUT is only used in a serial configuration daisy-chain for outputting data to the downstream FPGA (or for the BitGen DebugBitstream option). Otherwise, DOUT is high-Z.
4. CSO_B is only used in a parallel configuration daisy-chain for outputting a chip-enable signal to a downstream device. Otherwise, CSO_B is high-Z.
5. RS0 and RS1 are only driven when a MultiBoot event is initiated or when the BitGen ConfigFallback option is enabled and a Fallback event occurs. Otherwise, RS0 and RS1 are high-Z. When using the RS[1:0] pins for configuration it is recommended not to use them in User mode.
6. Empty cells indicate that the pin is not used in the configuration mode and is ignored and is high-Z during configuration.

The definition of each configuration pin is summarized in Table 2-4.

Table 2-4: **Configuration Pin Definitions**

Pin Name	Bank[1]	Type	Direction	Description
CFGBVS	0	Dedicated	Input	Configuration Banks Voltage Select CFGBVS determines the I/O voltage operating range and voltage tolerance for the dedicated configuration bank 0 and for the multi-function configuration pins in banks 14 and 15 in the AMD Spartan™-7, AMD Artix™-7 and AMD Kintex™-7 families. CFGBVS selects the operating voltage for the dedicated bank 0 at all times in all 7 series devices. CFGBVS selects the operating voltage for the multi-function configuration banks 14 and 15 only during configuration. Connect CFGBVS High or Low per the bank voltage requirements. If the V_{CCO_0} supply for bank 0 is supplied with 2.5V or 3.3V, then the CFGBVS pin must be tied High (i.e. connected to V_{CCO_0}). Tie CFGBVS to Low (i.e. connected to GND), only if the V_{CCO_0} for bank 0 is less than or equal to 1.8V. If used during configuration, banks 14 and 15 should match the VCCO level applied to bank 0. ***Caution!*** To avoid device damage, CFGBVS must be connected correctly to either V_{CCO_0} or GND. See Configuration Banks Voltage Select, page 28 for more information. ***Note:*** The CFGBVS pin is not available on AMD Virtex™-7 HT devices. Virtex 7 HT devices support only 1.8V operation for bank 0.
M[2:0]	0	Dedicated	Input	Configuration Mode M[2:0] determine the configuration mode. See Table 2-3, page 19 for the configuration mode settings. Connect each mode pin either directly, or via a ≤ 1 kΩ resistor, to V_{CCO_0} or GND.
TCK	0	Dedicated	Input	IEEE Std 1149.1 (JTAG) Test Clock Clock for all devices on a JTAG chain. Connect to AMD cable header's TCK pin. Treat as a critical clock signal and buffer the cable header TCK signal as necessary for multiple device JTAG chains. If the TCK signal is buffered, connect the buffer input to an external weak (e.g. 10 kΩ) pull-up resistor to maintain a valid High when no cable is connected.
TMS	0	Dedicated	Input	JTAG Test Mode Select Mode select for all devices on a JTAG chain. Connect to AMD cable header's TMS pin. Buffer the cable header TMS signal as necessary for multiple device JTAG chains. If the TMS signal is buffered, connect the buffer input to an external weak (e.g. 10 kΩ) pull-up resistor to maintain a valid High when no cable is connected.

Table 2-4: **Configuration Pin Definitions *(Cont'd)***

Pin Name	Bank[1]	Type	Direction	Description
TDI	0	Dedicated	Input	JTAG Test Data Input JTAG chain serialized data input. For an isolated device or for the first device in a JTAG chain, connect to AMD cable header's TDI pin. Otherwise, when the FPGA is not the first device in a JTAG chain, connect to the TDO pin of the upstream JTAG device in the JTAG scan chain.
TDO	0	Dedicated	Output	JTAG Test Data Output JTAG chain serialized data output. For an isolated device or for the last device in a JTAG chain, connect to AMD cable header's TDO pin. Otherwise, when the FPGA is not the last device in a JTAG chain, connect to the TDI pin of the downstream JTAG device in the JTAG scan chain.
PROGRAM_B	0	Dedicated	Input	Program (bar) Active-Low reset to configuration logic. When PROGRAM_B is pulsed Low, the FPGA configuration is cleared and a new configuration sequence is initiated. Configuration reset initiated upon falling edge, and configuration (i.e. programming) sequence begins upon the following rising edge. Connect PROGRAM_B to an external ≤ 4.7 kΩ pull-up resistor to V_{CCO_0} to ensure a stable High input, and recommend push-button to GND to enable manual configuration reset. ***Note:*** Holding PROGRAM_B Low from power-on does not keep the FPGA configuration in reset. Instead, use INIT_B to delay the power-on configuration sequence.
INIT_B	0	Dedicated	Bidirectional (open-drain)	Initialization (bar) Active-Low FPGA initialization pin or configuration error signal. The FPGA drives this pin Low when the FPGA is in a configuration reset state, when the FPGA is initializing (clearing) its configuration memory, or when the FPGA has detected a configuration error. Upon completing the FPGA initialization process, INIT_B is released to high-Z at which time an external resistor is expected to pull INIT_B High. INIT_B can externally be held Low during power-up to stall the power-on configuration sequence at the end of the initialization process. When a High is detected at the INIT_B input after the initialization process, the FPGA proceeds with the remainder of the configuration sequence dictated by the M[2:0] pin settings. Connect INIT_B to a ≤ 4.7 kΩ pull-up resistor to V_{CCO_0} to ensure clean Low-to-High transitions.

Table 2-4: **Configuration Pin Definitions *(Cont'd)***

Pin Name	Bank[1]	Type	Direction	Description
DONE	0	Dedicated	Bidirectional	Done A High signal on the DONE pin indicates completion of the configuration sequence. The DONE output is an open-drain output by default. ***Note:*** DONE has an internal pull-up resistor of approximately 10 kΩ. There is no setup/hold requirement for the DONE register. These changes, along with the DonePipe register software default, eliminate the need for the DriveDONE driver-option. External 330Ω resistor circuits are not required but can be used as they have been in previous generations.
CCLK	0	Dedicated	Input or Output	Configuration Clock CCLK runs the synchronous FPGA configuration sequence in all modes except JTAG mode. • For slave modes: CCLK is an input and requires connection to an external clock source. • For master modes: The FPGA sources the configuration clock and drives CCLK as an output. • For JTAG mode: CCLK is high-Z and can be left unconnected. ***Note:*** Treat CCLK as a critical clock signal to ensure good signal integrity (see the Signal Integrity page on xilinx.com for more information).
PUDC_B	14	Multi-function	Input	Pull-Up During Configuration (bar) Active-Low PUDC_B input enables internal pull-up resistors on the SelectIO pins after power-up and during configuration. • When PUDC_B is Low, internal pull-up resistors are enabled on each SelectIO pin. • When PUDC_B is High, internal pull-up resistors are disabled on each SelectIO pin. PUDC_B must be tied either directly, or via a ≤ 1 kΩ to V_{CCO_14} or GND. When PUDC_B is tied to GND, the activation of internal pull-ups during power-on depends on the power sequence because the PUDC_B control signal is forwarded through an input buffer in bank 14 and internal paths to the enables of internal pull-ups at applicable pins in their respective I/O banks. An external pull-up resistor is recommended between a pin and the pin's V_{CCO} power supply when it is critical for the pin to be pulled High immediately as the pin's V_{CCO} power ramps up. ***Caution!*** Do not allow this pin to float before and during configuration.

Table 2-4: **Configuration Pin Definitions *(Cont'd)***

Pin Name	Bank[1]	Type	Direction	Description
EMCCLK	14	Multi-function	Input	External Master Configuration Clock Optional external clock input for running the configuration logic in a master mode (versus the internal configuration oscillator). See Setting Configuration Options in the Vivado Tools, page 32 for more information. • For master modes: The FPGA can optionally switch to EMCCLK as the clock source, instead of the internal oscillator, for driving the internal configuration engine. The EMCCLK frequency can optionally be divided via a bitstream setting (ExtMasterCclk_en) and is forwarded for output as the master CCLK signal. • For JTAG and slave modes: EMCCLK is ignored in the JTAG and slave modes and can be left unconnected.
CSI_B	14	Multi-function	Input	Chip Select Input (bar) Active-Low input that enables the FPGA SelectMAP configuration interface. • For master SelectMAP mode: Connect CSI_B directly, or via a $\leq$ 1 kΩ resistor, to GND. • For slave SelectMAP mode: An external configuration controller can control CSI_B for selecting the active FPGA on the SelectMAP bus, or in a parallel configuration daisy-chain, connect to the CSO_B pin of the upstream FPGA. • In all other modes: CSI_B is ignored and can be left unconnected.
CSO_B	14	Multi-function	Output (open-drain)	Chip Select Output (bar) Active-Low open-drain output that can drive Low to enable the slave SelectMAP configuration interface of the downstream FPGA in a parallel configuration daisy-chain. • For BPI (asynchronous read only) and SelectMAP modes: If the device is in a parallel configuration daisy-chain and has a downstream device, then connect to an external 330Ω pull-up to V_{CCO_14} and connect to the CSI_B input of the downstream device. Otherwise, CSO_B is high-Z. • For serial modes: CSO_B is a multi-purpose pin that functions as the DOUT pin. See DOUT row in this table. • For all other modes: CSO_B is high-Z and can be left unconnected.

Table 2-4: **Configuration Pin Definitions *(Cont'd)***

Pin Name	Bank[1]	Type	Direction	Description
DOUT	14	Multi-function	Output	Data Output DOUT is the data output for a serial configuration daisy-chain. DOUT is clocked out on the falling edge of CCLK. • For serial and SPI (x1 only) modes: If the device is in a serial configuration daisy-chain, then connect to the DIN of the downstream slave-serial FPGA. Otherwise, DOUT is high-Z. • For BPI and SelectMAP modes: DOUT is a multi-purpose pin that functions as the CSO_B pin. See CSO_B row in this table. • For all other modes: DOUT is high-Z and can be left unconnected. ***Note:*** DOUT can output data when the DebugBitstream option is enabled.
RDWR_B	14	Multi-function	Input	Read/Write (bar) RDWR_B determines the direction of the SelectMAP data bus. When RDWR_B is High, the FPGA outputs read data onto the SelectMAP data bus. When RDWR_B is Low, an external controller can write data to the FPGA through the SelectMAP data bus. • For master SelectMAP mode: Connect RDWR_B directly, or via a ≤ 1 kΩ resistor, to GND. • For slave SelectMAP mode: An external device controls the RDWR_B signal to control the direction of the SelectMAP data bus for read/write from/to the SelectMAP interface. • In all other modes: RDWR_B is ignored and can be left unconnected.
D00_MOSI	14	Multi-function	Bidirectional	Master-Output, Slave-Input FPGA (master) SPI mode output for sending commands to the SPI (slave) flash device. • For SPI mode: Connect to the SPI flash data input pin. The D00_MOSI pin is high-Z after the command and address is sent to the SPI flash device. The PUDC_B pin determines if the signal will be pulled up. • For BPI and SelectMAP modes: The MOSI pin is a multi-purpose pin that functions as the D00 data input pin. See D[00-31] row in this table. • For all other modes: The MOSI pin function is not applicable, the pin is high-Z during configuration, is ignored during configuration, and can be left unconnected.

Table 2-4: **Configuration Pin Definitions *(Cont'd)***

Pin Name	Bank[1]	Type	Direction	Description
D01_DIN	14	Multi-function	Bidirectional	Data Input DIN is the serial data input pin. By default, data from DIN is captured on the rising edge of CCLK. • For serial and SPI modes: DIN is the FPGA data input that receives serial data from the data source. Connect DIN to the serial data output pin of the serial data source. • For BPI and SelectMAP modes: The DIN pin is a multi-purpose pin that functions as the D01 data input pin. See D[00-31] row in this table. • For JTAG mode: DIN is ignored.

Table 2-4: **Configuration Pin Definitions *(Cont'd)***

Pin Name	Bank[1]	Type	Direction	Description

D[00-31]	14	Multi-function	Bidirectional	Data Bus
				A subset or all of the D[00-31] pins are the data bus interface for the SPI x2, SPI x4, BPI, or SelectMAP modes. By default, data from the data bus is captured on the rising edge of CCLK.
				• For SPI mode: Configuration begins with the D00/MOSI and D01 pins of the data bus used for standard SPI (x1) serial data output and data input. Bitstream options can switch the SPI flash read mode to dual output (x2) or quad output (x4) modes.
				  • For SPI x1/x2/x4: Connect D00/MOSI to the SPI flash serial data input (DQ0/D/SI/IO0) pin.
				  • For SPI x1/x2/x4: Connect D01/DIN to the SPI flash serial data output (DQ1/Q/SO/IO1) pin.
				  • For SPI x4: Connect D02 to the SPI flash quad data bit 2 output (DQ2/W#/WP#/IO2) pin and connect to an external 4.7kohm pull-up resistor to VCCO_14.
				  • For SPI x4: Connect D03 to the SPI flash quad data bit 3 output (DQ3/HOLD#/IO3) pin and connect to an external 4.7kohm pull-up resistor to VCCO_14.
				The remaining data pins are unused, ignored, and high impedance during configuration.
				• For SelectMAP modes: The FPGA monitors the D[00-07] for an auto-bus-width-detect pattern that determines whether only D[00-07] (x8 bus width) are used or a wider (x16 or x32) data bus width is used. Connect used data bus pins to the corresponding data pins on the data source.
				Caution! The slave SelectMAP x16 and x32 data bus widths do not support configuration from AES-encrypted bitstreams.
				• For BPI mode: The FPGA monitors the D[00-07] for an auto-bus-width-detect pattern that determines whether only D[00-07] (x8 bus width) are used or a wider (x16) data bus width is used. Connect used data bus pins to the corresponding data pins on the BPI flash. The D[16-31] pins are multi-purpose pins that function as the BPI address A[00-15] pins. See A[00-28] row in this table.
				• For JTAG mode: None of the data pins are used.
				• For all modes: The unused data pins are high-Z and ignored during configuration. The unused data pins can be left unconnected.

Table 2-4: **Configuration Pin Definitions *(Cont'd)***

Pin Name	Bank[1]	Type	Direction	Description
A[00-28]	14 or 15	Multi-function	Output	Address Bus A[00-28] pins output addresses to a parallel NOR (BPI) flash. A00 is the least-significant address bit. • For BPI mode: Connect the FPGA A[00-28] pins to the parallel NOR flash address pins with the FPGA A00 pin connected to the least-significant flash address input pin that is valid for the used data bus width. Depending on the BPI flash type and used data bus width, the least-significant address bit of the flash can be A1, A0, or A-1. Note that any upper address pins that exceed the address bus width of the parallel NOR flash are driven during configuration, but can be used as I/O after configuration. • For SelectMAP mode: The A[00-15] pins are multi-purpose pins that function as the D[16-31] data bus pins. See D[00-31] row in this table. • In the other modes: A[00-28] are high-Z, are ignored during configuration, and can be left unconnected.
FCS_B	14	Multi-function	Output	Flash Chip Select (bar) Active-Low chip select output that enables SPI or BPI flash devices for configuration. • For SPI and BPI modes: Connect the FPGA FCS_B to the flash device chip-select input and connect to an external ≤ 4.7 kΩ pull-up resistor to V_{CCO_14}. • For all other modes: FCS_B is high-Z and can be left unconnected.
FOE_B	15	Multi-function	Output	Flash Output-Enable (bar) Active-Low output-enable control signal for a parallel NOR flash. • For BPI mode: Connect the FPGA FOE_B to the flash output-enable input and connect to an external ≤ 4.7 kΩ pull-up resistor to V_{CCO_15}. • For all other modes: FOE_B is high-Z and can be left unconnected.
FWE_B	15	Multi-function	Output	Flash Write-Enable (bar) Active-Low write-enable control signal for a parallel NOR flash. • For BPI mode: Connect the FPGA FWE_B to the flash write-enable input and connect to an external ≤ 4.7 kΩ pull-up resistor to V_{CCO_15}. • For all other modes: FWE_B is high-Z, and can be left unconnected.

Table 2-4: **Configuration Pin Definitions *(Cont'd)***

Pin Name	Bank[1]	Type	Direction	Description
ADV_B	15	Multi-function	Output	Address Valid (bar) Active-Low address valid output signal for a parallel NOR flash. • For BPI mode with flash that support an address valid input: Connect the FPGA ADV_B to the parallel NOR flash address valid input pin and connect to an external ≤ 4.7 kΩ pull-up resistor to V_{CCO_15}. For BPI mode with flash that do not support an address valid input: Do not connect the ADV_B pin. • For all other modes: ADV_B is high-Z, and can be left unconnected.
RS0, RS1	15	Multi-function	Output	Revision Select The RS0 and RS1 pins are revision selection output pins, intended to drive upper address lines on a parallel flash memory. Normally, RS0 and RS1 are high-Z during configuration. However, the FPGA can drive the RS0 and RS1 pins under two possible conditions. When the ConfigFallback option is enabled, the FPGA drives RS0 and RS1 Low during the fallback configuration process that follows a detected configuration error. When a user-invoked MultiBoot configuration is initiated, the FPGA can drive the RS0 and RS1 pins to a user-defined state during the MultiBoot configuration process. If fallback is disabled (default) and if MultiBoot is not used, or if SPI mode is used, then RS0 and RS1 are high-Z and can be left unconnected.
VCCBATT	N/A	Supply Voltage	N/A	Battery Backup Supply V_{CCBATT} is the battery backup supply for the FPGA's internal volatile memory that stores the key for the AES decryptor. For encrypted bitstreams that require the decryptor key from the volatile key memory area, connect this pin to a battery to preserve the key when the FPGA is unpowered. If there is no requirement to use the decryptor key from the volatile key storage area, connect this pin to GND or V_{CCAUX}. The pin name includes the "_0" bank designation but it is not an I/O and not affected by V_{CCO_0}.

Notes:

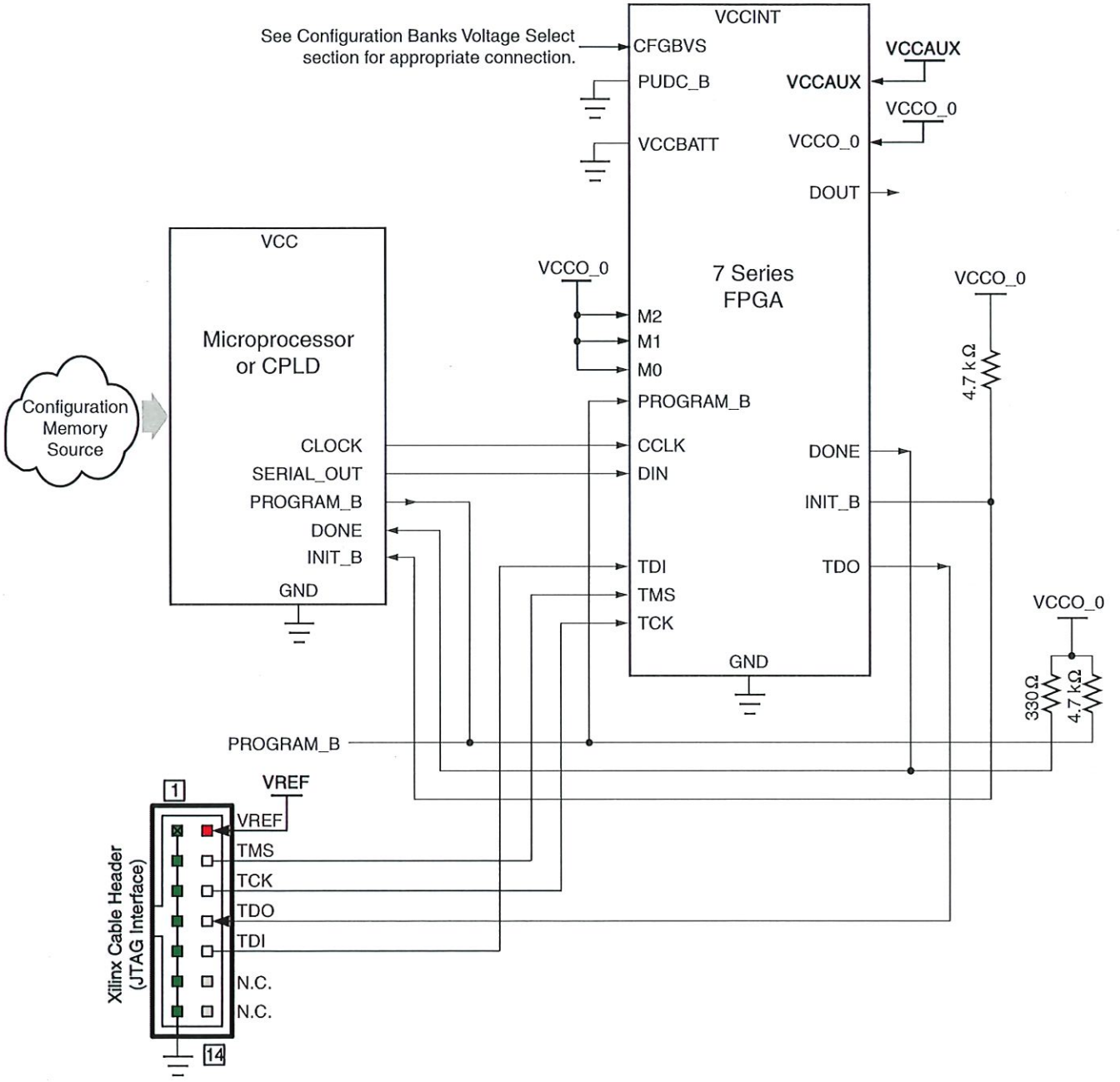
1. Each I/O is referenced to the V_{CCO} supply voltage for the bank in which the I/O is located. For example, "0" indicates the I/O is referenced to Bank 0's V_{CCO_0}.

Configuration Banks Voltage Select

The configuration banks voltage select (CFGBVS) pin must be set to High, or Low, in order to determine the I/O voltage support for the pins in bank 0, and for the multi-function pins in banks 14 and 15 when they are used during configuration. The CFGBVS is a logic input pin referenced between V_{CCO_0} and GND. When the CFGBVS pin is High (e.g., connected to the V_{CCO_0} supply of 3.3V or 2.5V), the configuration and JTAG I/O on bank 0 support operation at 3.3V or 2.5V during and after configuration. When the CFGBVS pin is Low (e.g., connected to GND), the I/O in bank 0 support operation at 1.8V or 1.5V. Configuration is not supported at 1.2V.

Slave Serial Configuration

Slave Serial configuration is typically used for devices in a serial daisy chain or when configuring a single device from an external microprocessor or CPLD (see Figure 2-2). Design considerations are similar to Master Serial configuration except for the direction of CCLK. CCLK must be driven from an external clock source, which also provides data (see Clocking Serial Configuration Data, page 35). For daisy chain information, see Chapter 9, Multiple FPGA Configuration.



Refer to the Notes following this figure for related information.

UG470_c2_02_021914

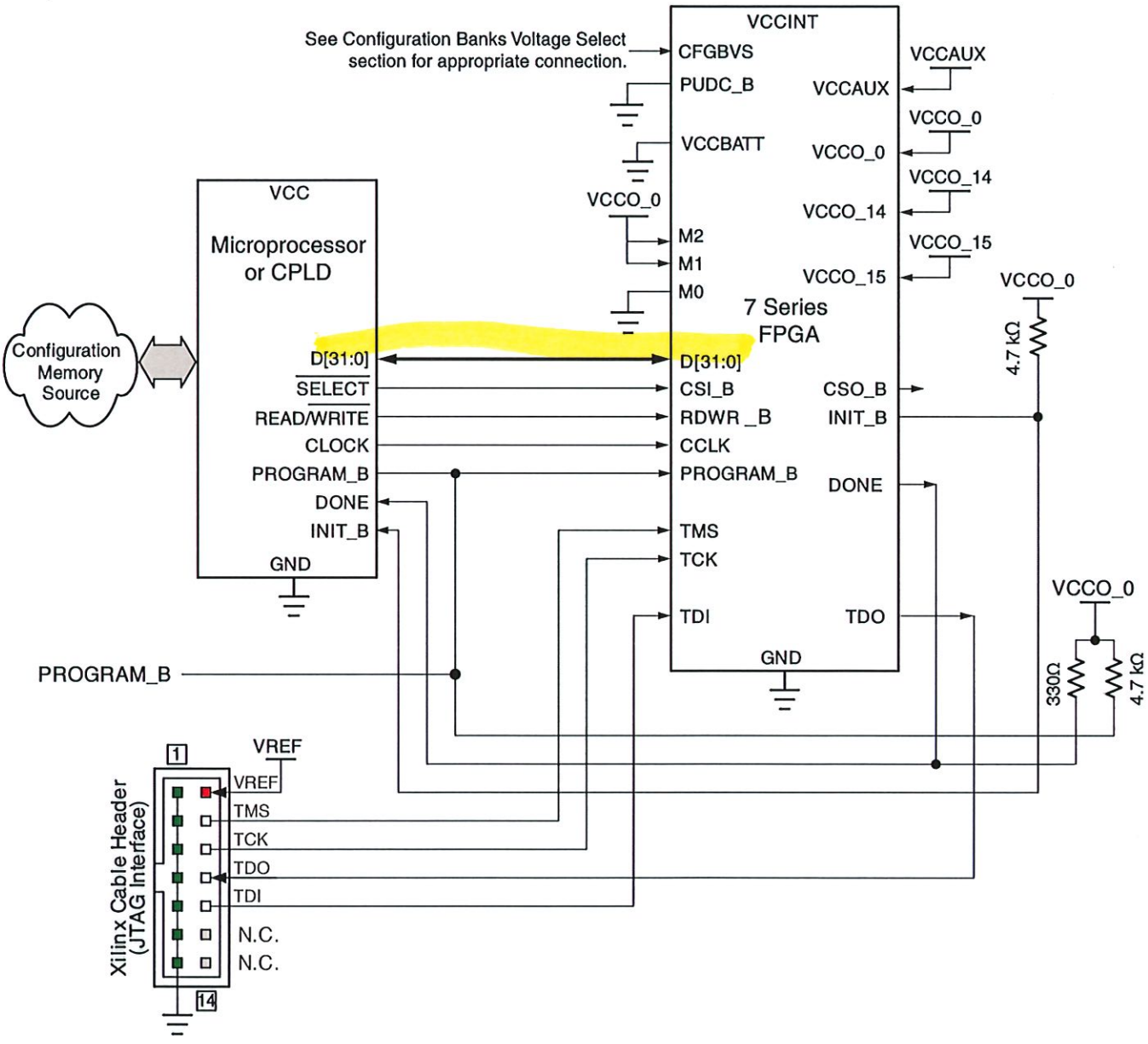
Figure 2-2: **Slave Serial Mode Configuration Example**

The SelectMAP configuration interface pins shown in Figure 2-4 are defined in Table 2-4, page 20.

Single Device SelectMAP Configuration

Microprocessor-Driven SelectMAP Configuration

For custom applications where a microprocessor or CPLD is used to configure a single 7 series device, either Master SelectMAP mode (use CCLK from the FPGA) or Slave SelectMAP mode can be used (Figure 2-5). Slave SelectMAP mode is preferred. See XAPP583, *Using a Microprocessor to Configure 7 Series FPGAs via Slave Serial or Slave SelectMAP Mode*, for information on configuring AMD FPGAs using a microprocessor.



Refer to the Notes following this figure for related information.

UG470_c2_05_072114

Figure 2-5: **Single Slave Device SelectMAP Configuration from Microprocessor or CPLD Example**

The ABORT sequence lasts four CCLK cycles. During those cycles, the status word changes to reflect data alignment and ABORT status. A typical sequence might be:

```
11011111  => DALIGN = 1,  IN_ABORT_B = 1
10001111  => DALIGN = 0,  IN_ABORT_B = 0
10001111  => DALIGN = 0,  IN_ABORT_B = 0
10001111  => DALIGN = 0,  IN_ABORT_B = 0
```

After the last cycle, the synchronization word can be reloaded to establish data alignment.

Resuming Configuration or Readback After an Abort

There are two ways to resume configuration or readback after an ABORT:

- The device can be resynchronized after the ABORT completes.
- The device can be reset by pulsing PROGRAM_B Low at any time.

To resynchronize the device, CSI_B must be deasserted then reasserted. Configuration or readback can be resumed by sending the last configuration or readback packet that was in progress when the ABORT occurred. Alternatively, configuration or readback can be restarted from the beginning.

Master SPI Configuration Mode

The 7 series FPGA Master SPI configuration mode enables the use of low pin-count, industry-standard SPI flash devices for bitstream storage. The FPGA supports a direct connection to the de facto standard, four-pin interface of a SPI flash device for reading a stored bitstream.

The 7 series FPGA Master SPI configuration mode (Figure 2-11) can optionally read from SPI devices that support x2 and x4 Fast Output Read operations. These output modes are proportionally faster than the standard 1-bit SPI interface. In addition, a negative edge clocking mode is available to make better use of the entire clock period and allow higher configuration speed. SPI flash densities over 128 Mb requiring 32-bit addressing are also supported. (See SPI Densities over 128 Mb, page 52 for additional instructions and limitations.) For information on using SPI configuration with 3D ICs, see 3D ICs Based on SSI Technology, page 14.

Figure 2-12, page 48 shows the connections for a SPI configuration with a x1 or x2 data width. These connections are the same because the x2 mode uses the D00_MOSI pin as a dual-purpose Data In/Out pin. Daisy-chained configuration mode is only available in SPI x1 mode. The FPGA pin connections to the SPI flash involved in the Master SPI mode are listed in Table 2-4, page 20.

The programming software provides the ability to program a SPI serial flash using an indirect programming method. This downloads a new FPGA design that provides a connection from the software through the 7 series device to the SPI flash. Previous FPGA memory contents are lost during this operation. For SPI flash supported by the Vivado tools, see UG908, *Vivado Design Suite User Guide: Programming and Debugging*. For the specific devices supported by the AMD ISE™ iMPACT programming tools, please consult the iMPACT Help documentation.

Note: The UG908 and iMPACT Help display supported flash devices however named devices are not an assurance of flash device availability. AMD recommends contacting your flash supplier for device availability.

For additional details on the SPI x1, x2, and x4 operation, including reference schematics and programming instructions, see XAPP586, *Using SPI Flash with 7 Series FPGAs*.

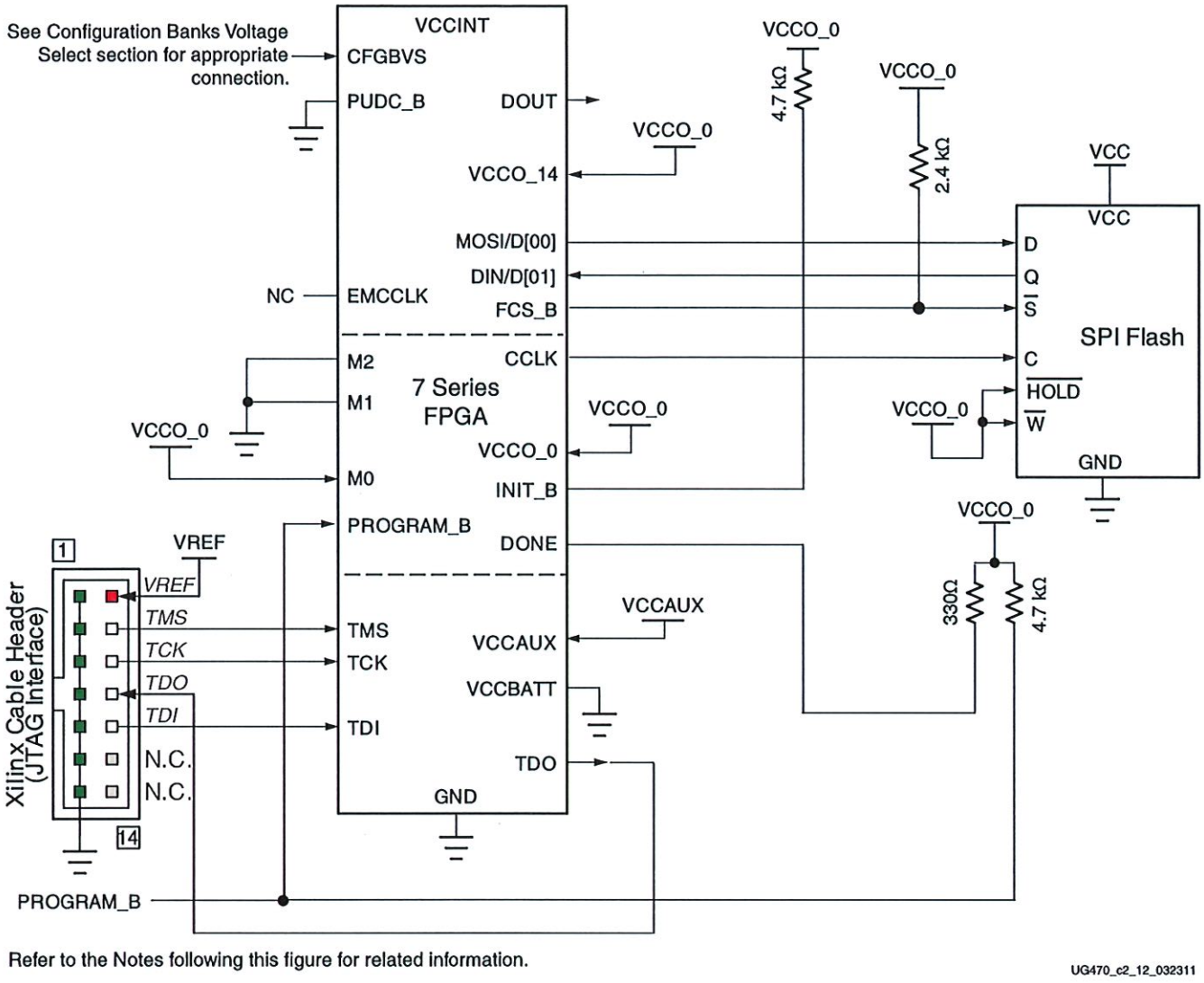


Figure 2-12: **7 Series FPGA SPI x1/x2 Configuration Interface**

Notes relevant to Figure 2-12:

1. The DONE pin is an open-drain output. See Table 2-4, page 20 for DONE signal details.
2. The INIT_B pin is a bidirectional, open-drain pin. An external pull-up resistor is required.
3. The bitstream startup clock setting must be set for CCLK for SPI configuration.
4. CCLK signal integrity is critical and termination might be required. Simulation is recommend to determine the appropriate termination as it is application dependent.
5. DOUT should be connected to the DIN of the downstream FPGA for daisy-chained SPI x1 configuration mode. Daisy-chaining is not supported for x2 or x4 SPI modes.
6. A series resistor should be considered for the datapath from the flash to the FPGA to minimize overshoot. The proper resistor value can be determined from simulation.
7. The 7 series FPGA V_{CCO_0} supply must be compatible with the V_{CC} for the I/O of the SPI device.
8. Data is clocked out of the SPI on the CCLK falling edge and clocked in on the FPGA on the rising edge, unless negative edge clocking (spi_fall_edge:Yes) is enabled.

- Hold the FPGA INIT_B pin Low from power-up to delay the start of the FPGA configuration procedure. Release the INIT_B pin to High after the SPI flash becomes ready to receive commands.

Master BPI Configuration Interface

The 7 series FPGA Master BPI configuration mode (Figure 2-16) enables the use of industry-standard parallel NOR (BPI) flash devices for bitstream storage. The FPGA supports a direct connection to the address, data, and control signals of a BPI flash for extracting a stored bitstream.

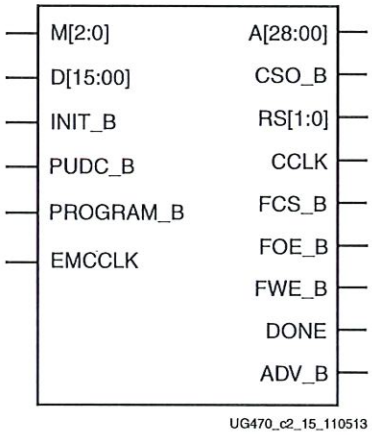


Figure 2-16: **7 Series FPGA Master BPI Configuration Interface**

The BPI configuration interface pins shown in Figure 2-16 are defined in Table 2-4, page 20. Note that some of the required pins are in bank 15. The CPG236 package for the Artix 7 7A50T and smaller devices does not bond out bank 15 and therefore does not support BPI configuration. The Spartan 7 family does not support BPI configuration.

The 7 series FPGA Master BPI configuration mode has two BPI flash read modes available: asynchronous and synchronous. Faster configuration times can be achieved using the BPI flash synchronous read mode with the external master clock than in other direct configuration modes. In addition, a wider density range of parallel NOR flash can be accessed by up to 29 address lines.

By default, 7 series FPGAs use the asynchronous mode of the BPI flash to read bitstream data as shown in Figure 2-17, page 56. The FPGA drives the address bus from a given start address, and the BPI flash sends back the bitstream data. The default start address is address 0. The start address can be explicitly set in a MultiBoot reconfiguration procedure as outlined in Chapter 7, Reconfiguration and MultiBoot. In asynchronous read mode, supported bus widths of x8 and x16 are auto-detected as described in Bus Width Auto Detection, page 74.

The 7 series FPGA Master BPI configuration mode can optionally read a bitstream from select BPI devices that support burst, synchronous read mode as illustrated in Figure 2-20 and described in Synchronous Read Mode Support, page 59. A BPI device that supports the synchronous read mode latches a given start address from the FPGA into its internal address counter. Then given a clock, the flash outputs data from the next sequential address location to its data bus during each clock period. In the synchronous read mode, a BPI device can deliver data many times faster than through its asynchronous read interface. Refer to the specific FPGA for the number of address signals available that determine the maximum flash density supported for configuration.

The programming software provides the ability to program parallel NOR flash using an indirect programming method. This method downloads a new FPGA design that provides a connection from the software through the 7 series FPGA to the BPI flash device. For more details, see XAPP587, *BPI*

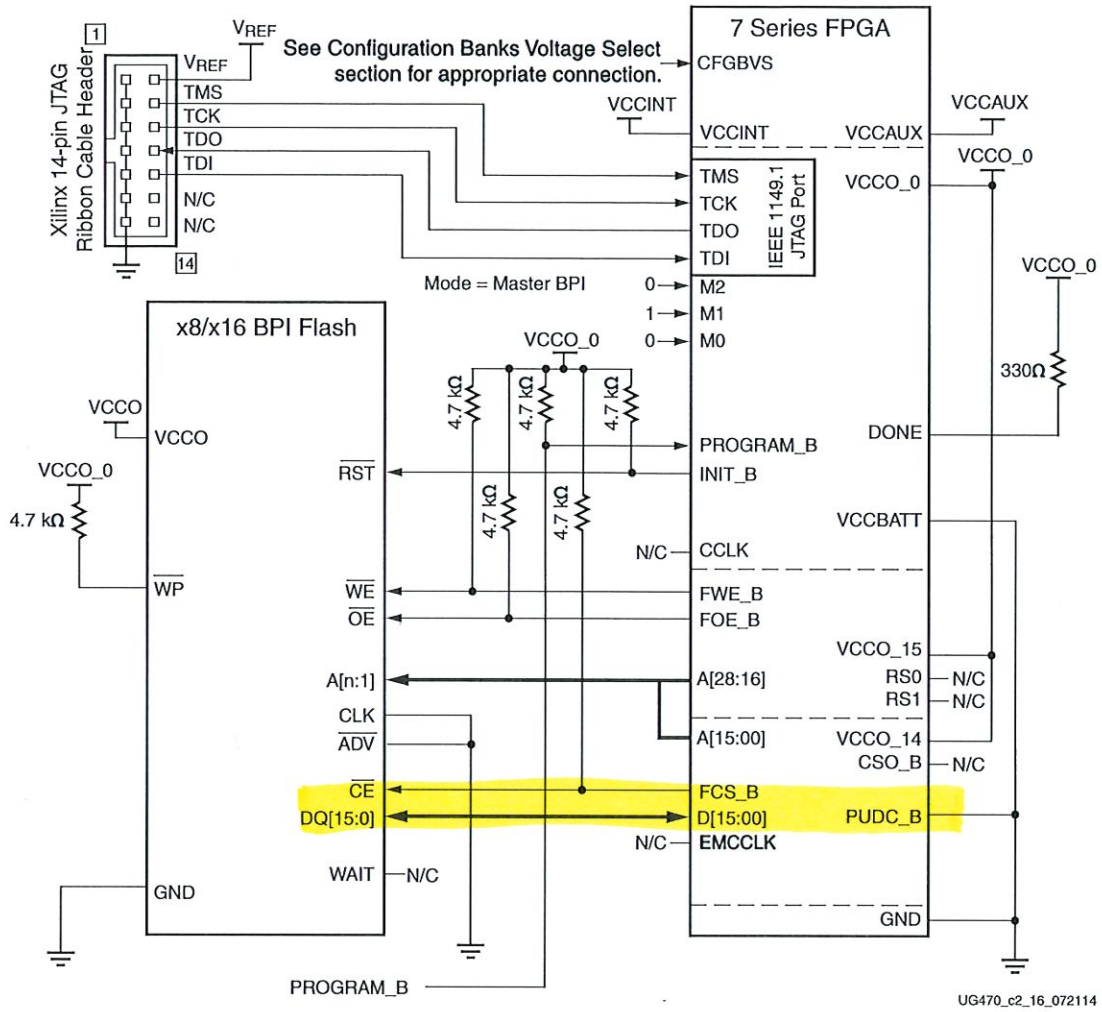


Figure 2-17: **7 Series FPGA Master BPI Configuration Interface - Asynchronous Read Example**

Notes relevant to Figure 2-17:

1. 7 series FPGA V_{CCO_0} supply input and AMD cable V_{REF} must be tied to the same voltage.
2. 7 series FPGA bank voltage V_{CCO_14} supplies: A[15:00], FCS_B, D[15:00], EMCCLK, PUDC_B, and CSO_B signals. Bank voltage V_{CCO_15} supplies: A[28:16], FWE_B, FOE_B, ADV_B, RS0, and RS1 signals.
3. M[2:0] = `010` for BPI mode.
4. Figure 2-17 shows the x16 BPI interface. For x8 BPI interfaces, only D[07:00] are used. See Bus Width Auto Detection, page 74.
5. Sending a bitstream to the data pin follows the same bit-swapping rule as in SelectMAP mode. See Parallel Bus Bit Order, page 77.
6. The CCLK output is not used to connect to flash in the asynchronous read mode, but it is used to sample flash read data during configuration. All timing is referenced to CCLK. The CCLK pin must not be driven or tied High or Low.
7. The RS[1:0] pins are not connected as shown in Figure 2-17. These output pins are optional and can be used for MultiBoot configuration. See Chapter 7, Reconfiguration and MultiBoot.
8. The DONE pin is an open-drain output. See Table 2-4, page 20 for DONE signal details.

The FPGA's master configuration clock has a tolerance of $T_{MCCKTOL}$. Due to the master configuration clock tolerance ($T_{MCCKTOL}$), the `ConfigRate` option must be checked so that the period for the worst-case (fastest) master CCLK frequency is greater than the sum of the FPGA address valid time, BPI flash access time, and FPGA setup time, as shown in Equation 2-2.

$$\frac{1}{ConfigRate \times (1 + FMCCKTOL_{MAX})} \geq T_{BPICCO} + T_{ACC} + T_{BPIDCC} \qquad \text{Equation 2-2}$$

Power-On Sequence Precautions

At power-on, the FPGA automatically starts its configuration procedure. When the FPGA is in a Master-BPI configuration mode, the FPGA asserts FCS_B Low and drives a sequence of addresses to read the bitstream from a BPI flash. The BPI flash must be ready for asynchronous reads before the FPGA drives FCS_B Low and outputs the first address to ensure the BPI flash can output the stored bitstream.

Because different power rails can supply the FPGA and BPI flash or because the FPGA and BPI flash can respond at different times along the ramp of a shared power supply, special attention to the FPGA and BPI flash power-on sequence or power-on ramps is essential. The power-on sequence or power supply ramps can cause the FPGA to awaken before the BPI flash, or vice versa. For many systems with near-simultaneous power supply ramps, the FPGA power-on reset time (T_{POR}) can sufficiently delay the start of the FPGA configuration procedure such that the BPI flash becomes ready before the start of the FPGA configuration procedure. In general, the system design must consider the effect of the power sequence, the power ramps, FPGA power-on reset time, and BPI flash power-on reset time on the timing relation between the start of FPGA configuration and the readiness of the BPI flash for asynchronous reads. Check the respective 7 series FPGAs data sheet for 7 series FPGA power supply requirements and timing. One of these system design approaches can ensure that the BPI flash is ready for asynchronous reads before the FPGA starts its configuration procedure:

- Control the sequence of the power supplies such that the BPI flash is certain to be powered and ready for asynchronous reads before the FPGA begins its configuration procedure.
- Hold the FPGA INIT_B pin Low from power-up to delay the start of the FPGA configuration procedure and release the INIT_B pin to High after the BPI flash becomes ready for asynchronous reads.

JTAG Interface

From the four-pin JTAG interface, the 7 series FPGA can be configured using AMD tools and a AMD cable, directly from a processor or CPLD customer-specific design, or using third-party boundary-scan tools. The JTAG specific mode setting is (M[2:0] = `101`). AMD tools use the JTAG interface, including ISE and the AMD ChipScope™ Pro tool in the ISE Design Suite, and in the Vivado Design Suite lab tools.

Although JTAG commands have priority over mode settings, it is recommended to have an M[2:0] option to enable the JTAG mode and operations without potential conflict from other configuration modes. For more information, refer to Chapter 3, Boundary-Scan and JTAG Configuration.