

# LED Blinky Tutorial

FPGA's and SoC can be used for many more advanced functions as making a LED to blink. However "LED Blinky" is considered a "Hello World" for systems that can not otherwise say Hello.

## Requirements

To make LED to Blink (under user control) following requirements are needed:

1. Some Programmable Device: MCU, CPU, CPLD, FPGA, SoC
2. Working Programming or Configuration interface or means to boot the Programmable Device
3. Timing Source (Clock), can be external or internal, must be be enabled
4. Active Power to all required power domains of the Programmable Device
5. Release of main Reset
6. LED Connected to some user controllable I/O Port of the Programmable Device
7. LED must be assembled with right polarity and proper current limiting resistor
8. Valid I/O Voltage applied to the I/O Bank with the user LED



If any of those requirements are missing you may encounter problems with LED Blinky.

## Programmable Device

## Timing Source

A timing source or Clock is required to make an LED Blink, without reference to time or known Clock source, we can only make LED to be permanently ON or OFF.

If we do not know how to access the time/clock source or if we are not sure if the clock is enabled and running we have an potential problem and we can not proceed to the LED Blinky unless we have identified an active/enabled accessible clock source we can use for timing.

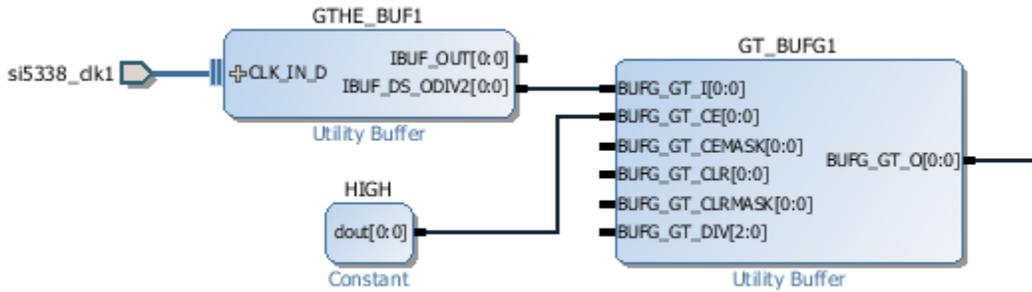
## FPGA Devices

### GT Clock

Gigabit Transceiver Clock may also be used as FPGA Fabric clock.



Clock buffers for GT Clock in Xilinx 7 Series Devices (example from TE0745 design)



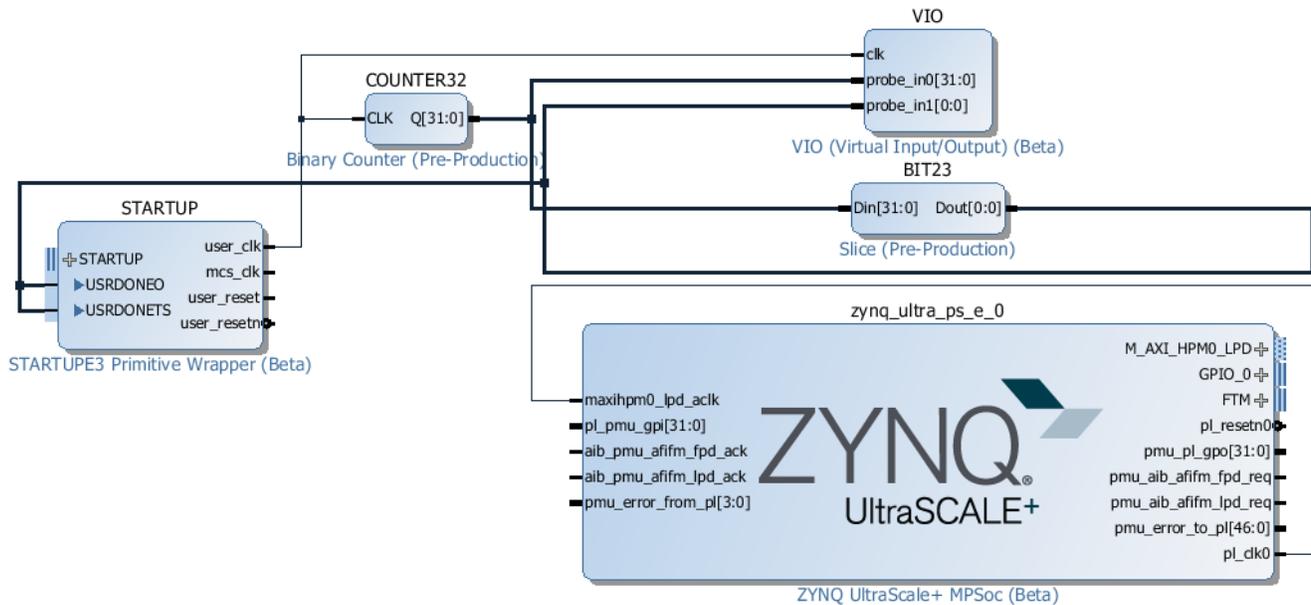
Clock buffers for GT Clock in Ultrascale Devices (example from TE0841 design)

## ZYNQ Devices

In Zynq System the main System Clock is connected PS (Processing Subsystem) and is not directly available to the PL (Programmable Logic - FPGA) unless the PS has enabled it during FSBL boot process. So if we have a generic Zynq Board then we can not expect to have clocks available to the FPGA until the Processing system has provided them. There may be clocks available to the PL that are active when the PS is not booted but this not a requirement.

If our goal is to make a LED to blink (from PL in Zynq Device), then the safest way is to use FPGA Configuration Master Clock this clock is always available and accessible in the same way, we do not need to know any specifics to the board we have and we do not depend on PS init done by FSBL.

## ZYNQ MPSoC Devices



This design blinks a LED connected to DONE pin and also a VIO Pin, on any Zynq MPSoC device, without using any clocks provided by the externally, and without the use of PS supplied clocks. LED Toggle rate is about 2 Hz.

## Valid I/O Voltage

It is very common that modern Integrated Circuits have separate power supply pins for I/O Voltage. That voltage if applied control the output levels from the device, if the Voltage is too low or missing then it is not possible to have enough current from the device I/O Pin to make a LED Lit.

FPGA and SoC SoM's can have fixed and user controlled Voltages for the I/O Banks. It is important to assure that the I/O Bank that controls the LED has a valid Supply Voltage applied.



NOTE: Missing VCCIO Voltage will not cause AXI GPIO to freeze, no matter if it is connected to MicroBlaze or Zynq Processing Block.



NOTE: Changing I/O Standard will not change the Bank Supply Voltage or enable it.

## DONE LED

If there is on board LED directly connected to FPGA "DONE" output, then it is also possible to control it as long as it is possible to configure the FPGA (or PL portion of Zynq). For Zynq the PS subsystem is not needed if configuration is done via JTAG.

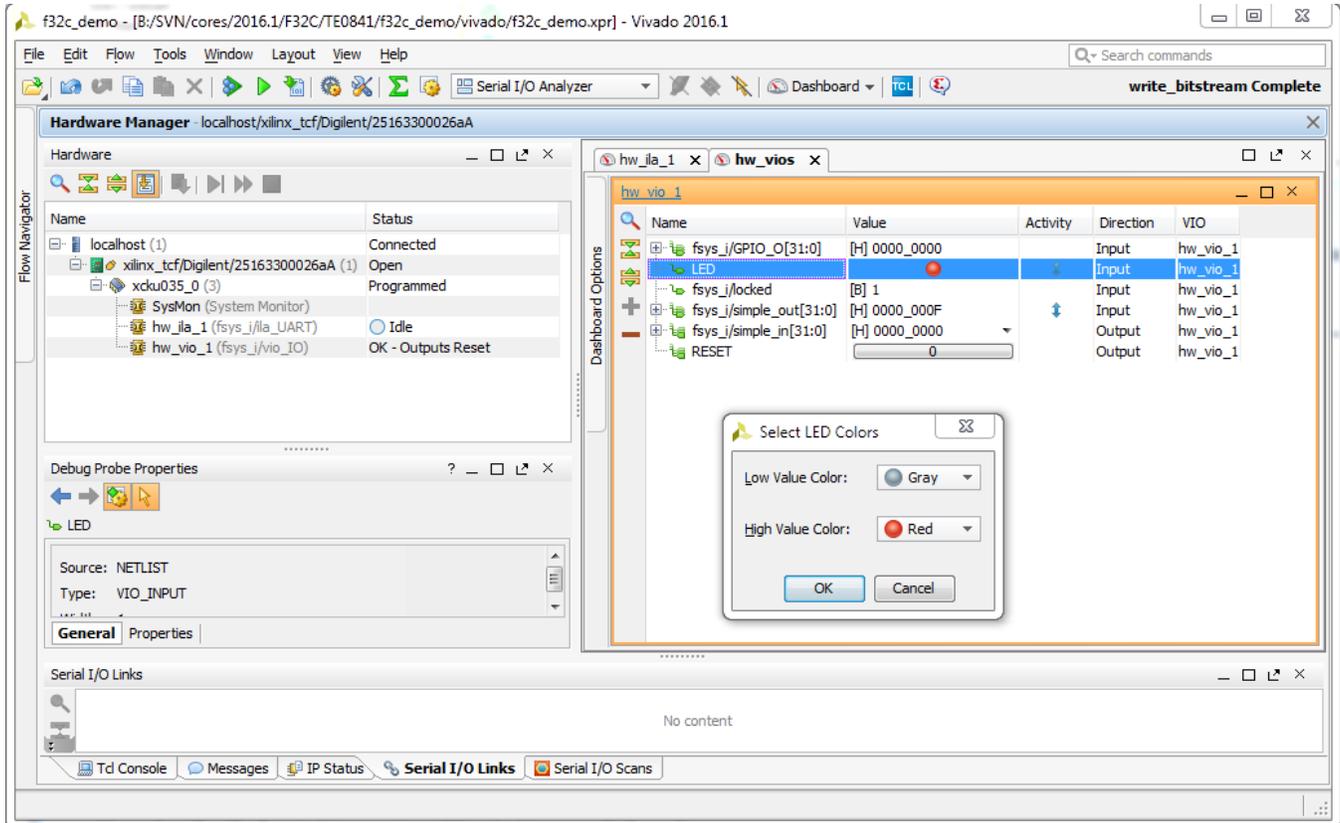
The good thing about Done LED is that if it is available it has fixed location, there is no need to assign it to some pin and it has fixed available power as well. The tricky parts is that DONE output is controlled by FPGA Primitive so the code to control the Done LED is different from that of normal PL I/O Controlled LED.

## MIO LED

If there is a LED directly connected to lower MIO Bank of the Zynq then this LED is also always controllable, as this Bank has to be powered for the device to boot. This LED can also be controlled if the PL portion of the Zynq is not configured or even un-powered.

## VIO LED

This is an option that is always available as long as we have JTAG Connection to the PL.



Example of VIO LED, the SoM being used is Kintex UltraScale TE0841, the LED is implemented in F32C Softcore Bootloader that is waiting for Arduino IDE to download a sketch. There is no limit for the number of virtual LED's. It is also possible to attach those VIO LED's to Zynq GPIO when using EMIO multiplexing, so the same GUI would show LED's controlled by Zynq PS during boot or running Linux.

## LED's on TE SoM's and Baseboards

	MIO LED	DONE LED	User LED PL	Notes
TE0710	-			
TE0711	-			
TE0712	-	No*	Yes*	LED's are routed to PL I/O by System Control CPLD
TE0714	-	No	Yes	One fixed LED on PL, 1.8V or 3.3V I/O Voltage always present

TE0720	MIO7	Yes	Yes*	System control CPLD can remap the LED functions during FSBL
TE0725	-	Yes	Yes	LED's power always available
TE0729	MIO7	Yes	-	
TE0741	-			
TE0745	MIO		-	
TE0782				
TE0841	-	-	Yes	

Base	User LED's	Notes
TE0701	up to 8 max	All LED's controlled by the System Control CPLD, some require user I/O Bank voltage
TE0703	2*	User supplied I/O Voltage
TE0705		
TE0706	0	Can use RJ45 LED's as PL User LED's
TEB0729		

## Blinky Examples

### MicroBlaze MCS

TBD

### MicroBlaze

TBD

## Arduino F32C

F32C is an Open Source Soft Processor and SoC system that has ready to use support for Arduino IDE (using Arduino IDE package manager). More details on installation into Arduino IDE can be found [here](#). F32C project is hosted at [github f32c/f32c](#).

#### Requirements

- Arduino IDE
- F32C-Arduino packages installed (this will pull in cross-compiler and libs)
- Supported SoM/Baseboard (see list in Table below)
- UART Connection
- JTAG Connection (or pre-programmed SoM)



```
Blink | Arduino 1.6.6
File Edit Sketch Tools Help
Blink$
// LED Blinky for FPGA LED Blinky Tutorial by Trenz Electronic
// the setup function runs once when you press reset or power the board
void setup() {
  // Not required for F32C simple output, those are always output
  pinMode(8, OUTPUT);
}
// the loop function runs over and over again forever
void loop() {
  digitalWrite(8, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(500); // wait for a half a second
  digitalWrite(8, LOW); // turn the LED off by making the voltage LOW
  delay(500); // wait for a half second
}
Done uploading.
cc1.exe: warning: command line option '-fpermissive' is valid for C++/ObjC++ but
Sketch uses 1,416 bytes (1%) of program storage space. Maximum is 129,536 bytes.
Invalid library found in C:\Users\anttil\AppData\Local\Arduino15\packages\FPGA\ardu
25 Generic FPGA board, MIPS, 128 KB BRAM Internal, 100 MHz, Hex 115.2 kbit/s no verify RS232, Generic on COM5
```

This is Arduino IDE screenshot after downloading LED Blinky sketch to F32C inside FPGA, selected generic FPGA board with 128K RAM and 100MHz Clock. After download the on-board LED on the FPGA/SoC SoM will blink.

## Troubleshooting Tips

If you are not sure to what pins the LED(s) are connected it is possible to create a "dummy" FPGA design with "UNUSED PIN PULLUP" option. If FPGA is loaded with bitfile then any LED that is connected in active high configuration will be on, if the I/O Bank is powered. The internal pullup current is sufficient to make most modern LED to be lit (not very bright).

If you know that there are LED's connected to FPGA and they do not get lit with "unused pin pullup" FPGA design, then most likely the I/O Bank with the LED's has no power.

