UART

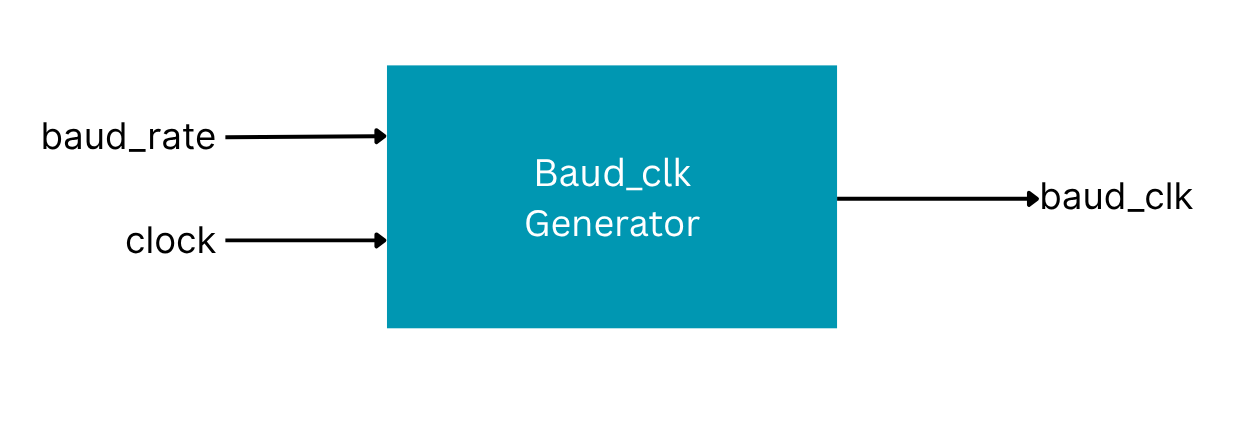
UART stands for Universal Asynchronous Receiver/Transmitter. It’s not a communication protocol like SPI and I2C, but a physical circuit in a microcontroller, or a stand-alone IC. A UART’s main purpose is to transmit and receive serial data.

It has 3 main circuits :- 1. Baud\_clk\_generator circuit

2. transmitter circuit

3. receiver circuit

1.Baud\_clk\_generator circuit :-



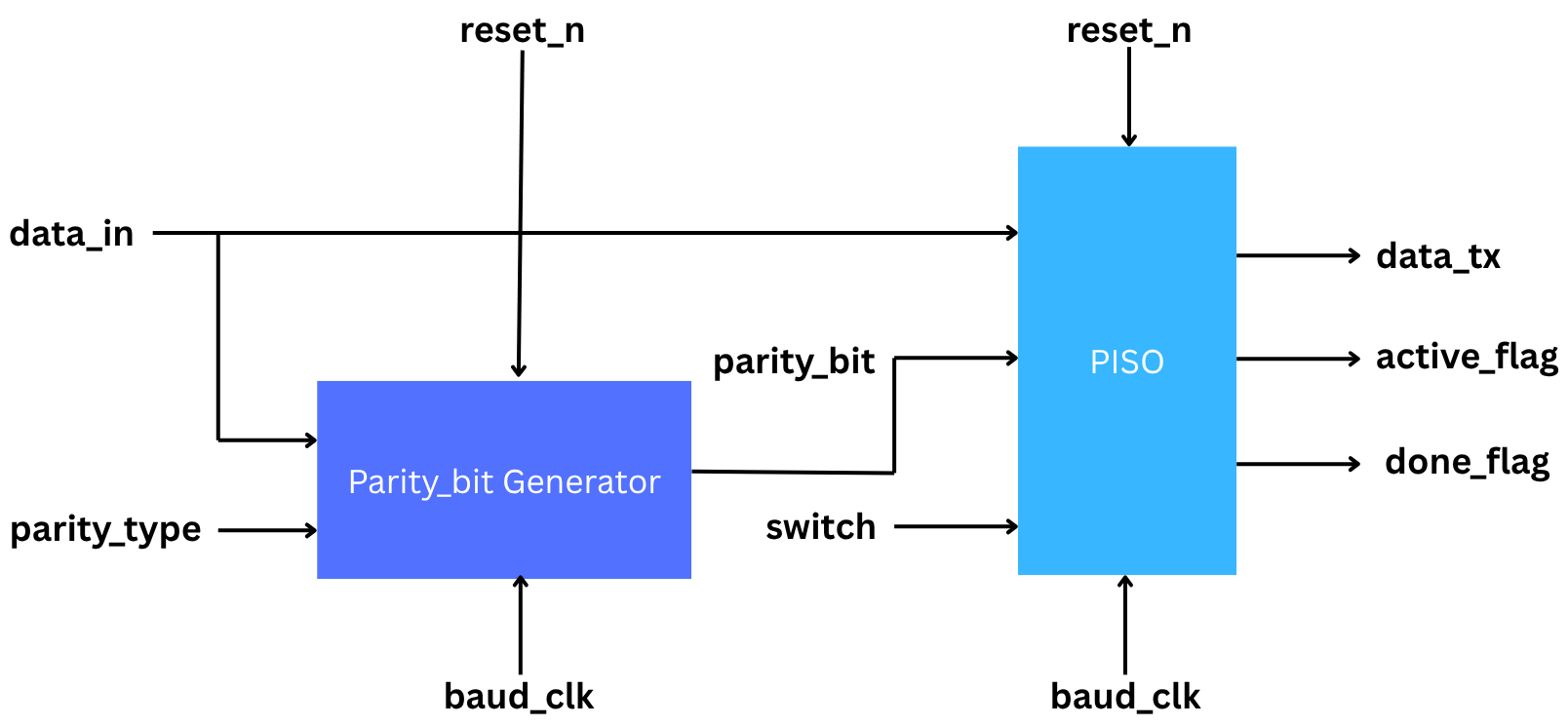
We know that FPGA has it’s own clock which is known as system’s clock , which is very fast , so data transimission with such a speed could lead to bit errors and jitter, therefore to make data transfer effiecient , we need to slow the clock , and the slower version of clock is known as baud\_clk.

Concept :- so to slow the clock specific number of clockedge replaced by one baud/clockedge

Formula :- bits/sec = clockfrequence/baudclock\*prefactor

There are four types of baud clock

2.transmitter circuit



It consists of 2 circuits :- 1. PISO 2. Parity\_bit Generator

1.PISO has 5 inputs and 3 outputs where it takes the data which has to be tranfered by data\_in , it starts the transmission when switch is ON , it receives parity bit from parity\_bit generator , which is useful for error detection .

The rate of transmission is decided by baud\_clk , and sudden stop of circuit is decided by reset\_n , when reset\_n is 0 then no matter which bit is being tranferrred , the system gets shut down immediately .

In PISO , in 8bit data :- start bit , parity bit and stop\_bit is added such thst it becomes 11bit frame .



Parity\_bit Generator produces parity bit , when number of 1 are odd so xor of data-in must be 1 , and paraity Type is also odd , then parity bit is 0 else 1 similarly for parity type = EVEN

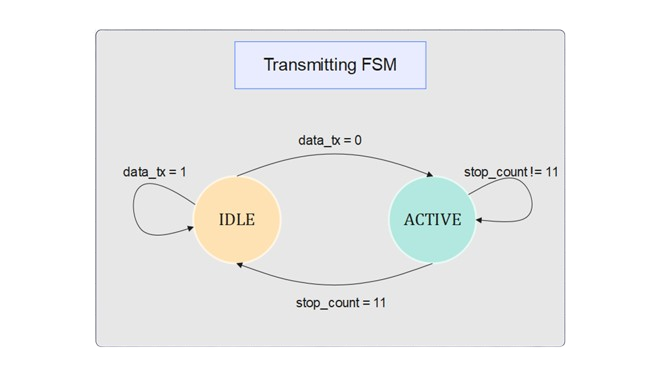
PISO has 3 outputs 1.data\_tx , 2.active flag 3.done\_flag

Data\_tx is a serially transmitted bit one by one through PISO circuit .

Active\_flag tells us whether the transmission is going on or not

Done\_flag tells us whether the transmission is done or not

Here, one FSM mechanism is being followed :-

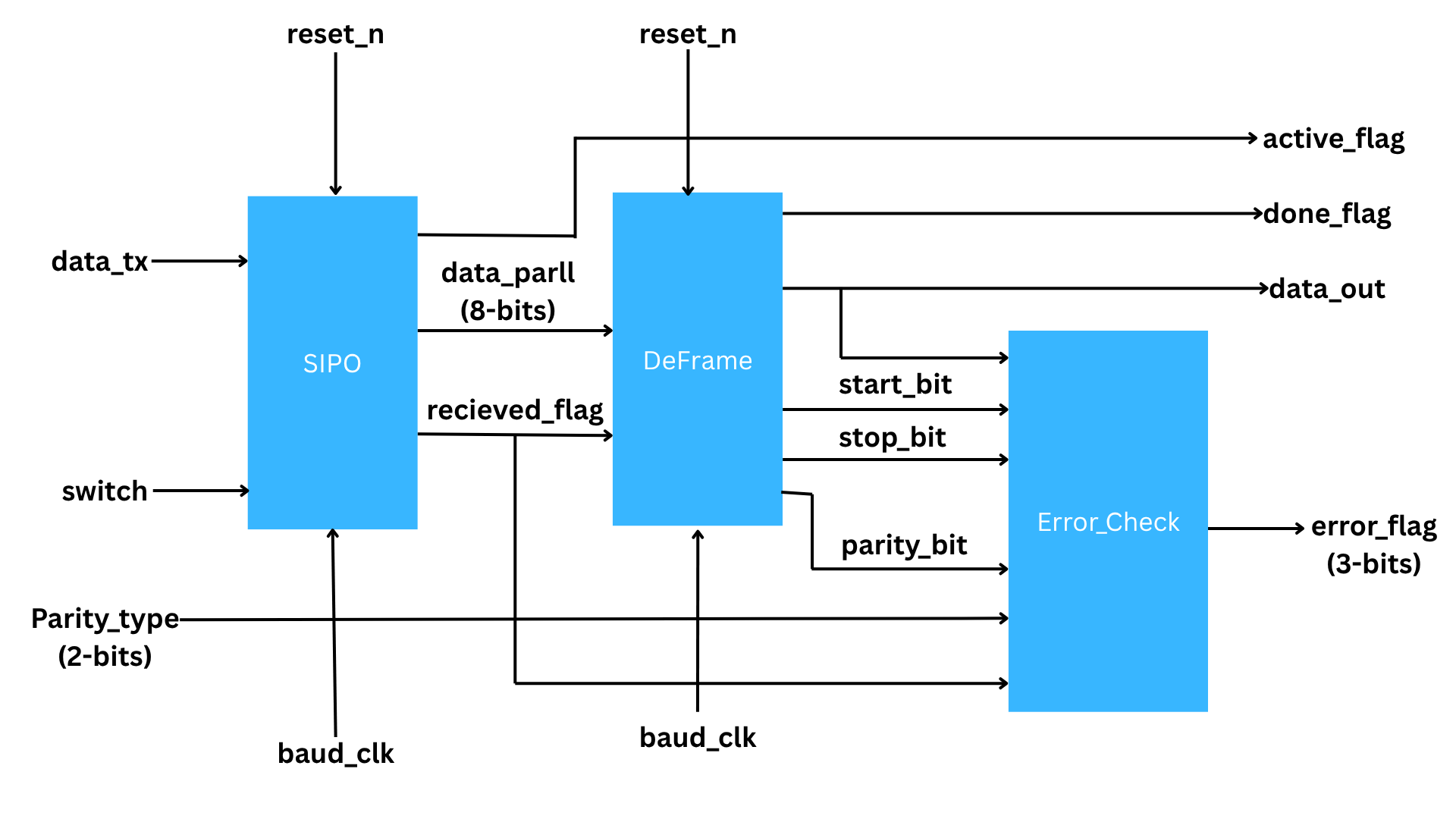
0

When the state is IDLE then system is in rest , when start bit arrives ,:- firstly whole transmission line is high , to start the transmission , we have to snatch down the line from high to low using start bit.

Then it goes to active state :- where it stop count is counter which counts upto 11 bits , when it counts upto 11 then count\_full is ON , and state changes to idle , done\_flag is ON and active \_flag is OFF.

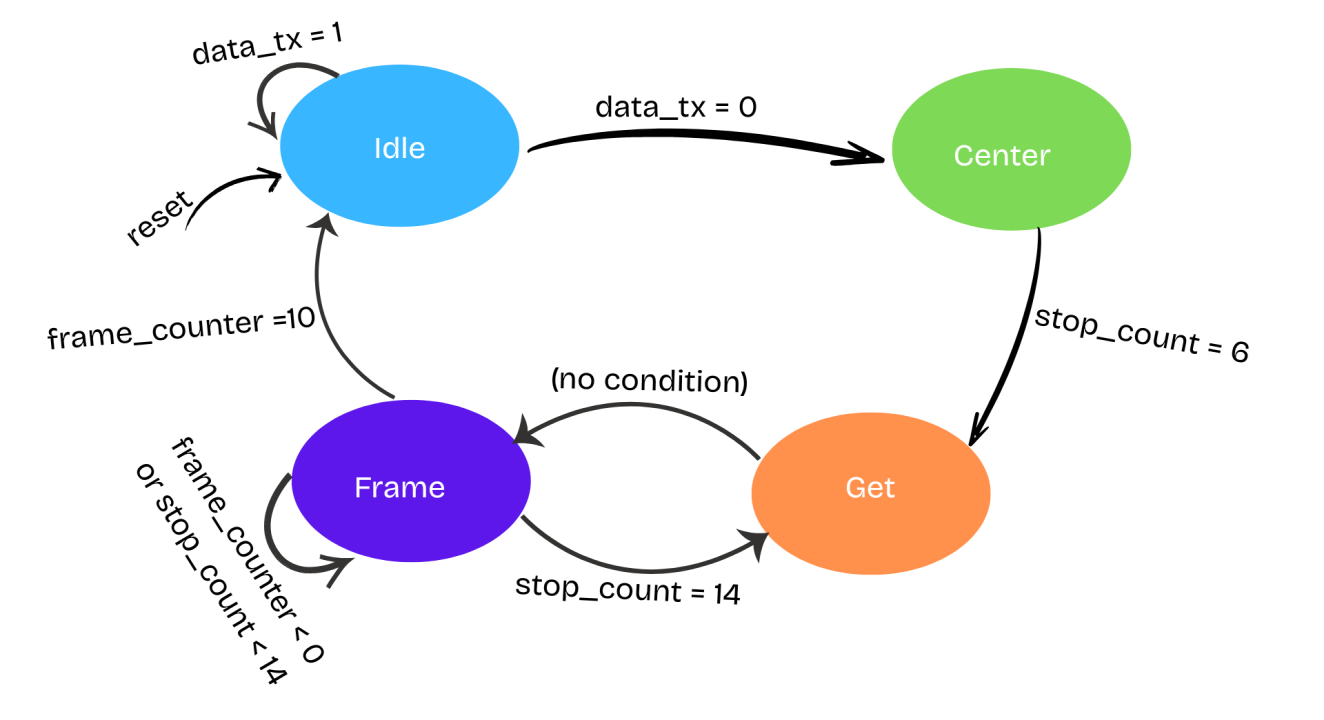
Hence the transmission part is done.

3.Receiver circuit :-



The data transferred by the transmitter circuit through data\_tx wire is received by receiver circuit serially , when switch is ON , then further the serial data is transferred from SIPO in form of parallel data through wire “data\_parll” to deframing it .

In SIPO ,one FSm mechanism is used :-



Functions in each stage :-

1.Idle :- all the quantities become zero like :- frame\_counter = 0 , stop\_count = 0 and transmission line become inactive , therefore frame becomes full 1 for 11 bit , because transmission line becomes on when it is put high to low from start bit.

2.Center :- stop\_count = stop +1

(the concept of using stop\_count is creating a delay , delay is most required is because

Important question :- why delay is nessecary , since we have already use the baud\_clk , which is the slower version of clk , so it provides the system enough time to execute the process?

**🕰️ Baud Clock vs. Sampling Timing**

**✅ What you're thinking:**

* "I already have a slow baud\_clk, so isn't that enough timing-wise to read bits one by one?"

Yes, **in theory**, you could sample on every rising edge of baud\_clk — **if** you're guaranteed that:

1. The serial transmitter and receiver are **perfectly aligned**, and
2. baud\_clk edges occur **right at the center** of each bit.

But in real-world digital systems and UART-style communication, this **alignment cannot be assumed**.

**❗ Why the delay (stop\_count) is necessary:**

**1. The falling edge of the start bit is asynchronous**

* Your FSM detects the start bit when data\_tx goes from 1 → 0.
* But this transition can occur **anywhere** with respect to your baud\_clk edge.
* So, if you immediately sample the next bit on the next baud\_clk, you might be **sampling too early or too late**.

📉 *Example:* The start bit might fall right **after** a baud clock edge. If you read the next bit 1 baud\_clk later, you'll be sampling **at the edge**, not center.

**2. To Sample in the Middle of a Bit**

For robust data recovery, bits are sampled in the middle of their period, where they are stable.

If you don't delay, you might sample during a transition (especially with noise or jitter), leading to bit errors.

➤ So, even though you're using a baud\_clk, you still need a delay (stop\_count) to:

Align to the center of the first bit (after start)

Space out subsequent reads properly to hit the center of every bit

Without stop\_count, problems may arise:

If you just read on every baud\_clk, and the initial detection was slightly off, every subsequent bit sample would also be off, possibly sampling during transitions.

Over time, this can accumulate errors, especially in noisy environments.

Conclusion: Why the Delay Is Necessary

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Reason | Explanation | | Start bit detection is asynchronous | It can happen anytime between baud\_clk edges | | Need center sampling | Sampling at the middle of each bit reduces bit errors | | Baud clock isn’t perfectly aligned | You can't assume baud\_clk edges are at the center of bits | | Robustness | Adds tolerance against jitter, clock drift, and metastability | |

So yes — baud\_clk sets the pace, but stop\_count aligns the sampling point precisely within that pace.

So , Center state positions the baud\_clk edge at the centre of bit , for errorless transfer.

3.Get :- This is the state where data from PISO is received to SIPO through data\_tx wire shifts the data one by one to the temp wire using shift register mechanism and then immediately state changes to Frame , without any condition .

4.Frame:- frame\_counter counts the number of bit being received by SIPO using

frame\_counter = frame\_counter + 1 at every recived bit , then it creates delay between two receiving bits using stop\_count , it changes to GET state when stop count becomes 14 , until then stop\_count = stop\_count + 1 , so 14 clock cylces delay is created after every bit received .

Structure look like this :-

Start bit detected

↓

Wait 6 cycles → Sample 1st data bit

↓

Wait 14 cycles → Sample 2nd data bit

↓

Wait 14 cycles → Sample 3rd data bit

...

↓

Done → Output 11-bit data

“Deframe” circuit seperates the the start\_bit , stop\_bit , and parity\_bit and data from the frame . it is on when full data is received which is detected by received\_flag , after the data is separated from the frame , done\_flag is on. Hence we get transferred data through data\_out.

“Errror\_Check” circuit detects the error in data using parity bit and parity\_type combination , so if the parity\_bit matches the uninary xor of data\_out then hence no error , but if does not matches thhe error is detected and error\_flag is ON .

|| Inn this way , our data is transmitted and received

through UART and how UART works||