GreenDroid Configuration Network with Clock Domain Crossing

A Bespoke Silicon Group Technical Paper

Bespoke Silicon Group

Department of Computer Science and Engineering

University of California, San Diego

February 25, 2013 - La Jolla

Abstract

In this technical paper we introduce the design and implementation of the configuration network for GreenDroid chip. Rather than use a traditional scan chain for this purpose, we develop an innovative way to broadcast configuration data while the chip is still running and let only the target node capture data and apply to its associated configurable pins. Since the network is likely to operate in a different clock domain than the rest of the GreenDroid chip, we also spend effort working on an efficient and reliable clock domain crossing technique.

Contents

1 Introduction

The design of GreenDroid chip includes multiple instances of embedded IPs such as PLLs, RAM controllers, bus interfaces and communication cores. I/O pins of those IPs need to be connected properly, sometimes without the intervention of the main processor. For example some PLL macro supports dynamic full PLL configuration via its input pins, and it is better and safer to use an independent configuration approach instead of using the main processor itself, to prevent system from halting in case of configuration fault.

Our design of the configuration network is to safely and easily program some embedded IP block pins of interest. Because configurations have to apply on-the-fly to programmable pins, some of which are critical (e.g., PLL pins), this design has to be competent in the follows.

- Consume minimum number of chip I/O pads
- Operate independently of the rest part of the chip
-

2 Design of the Configuration Network

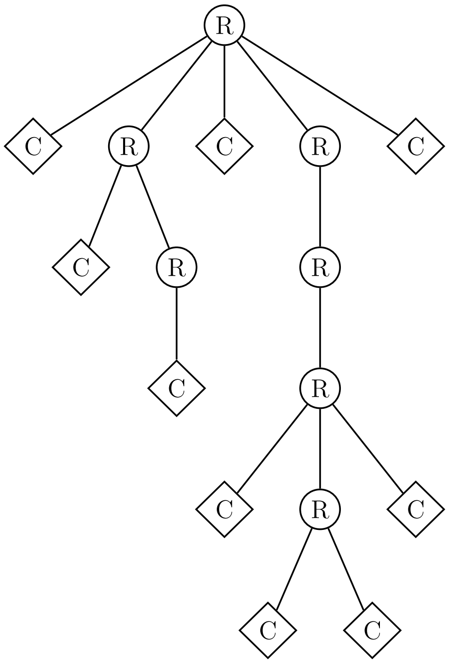

Figure 1: Configuration Network for GreenDroid
R indicates a relay node and C indicates a config node

Figure 1 illustrates an instance of the configuration network for GreenDroid. The normal structure of this network is a tree spanned by relay nodes, and one config node is connected as a leaf to a relay node which is geographically close to the pins to be configured. Compared to a traditional and linear scan chain, our design is more flexible in topology and able to follow the system hierarchy.

The input to the root relay node of the tree is connected to the chip I/O pads. Thereafter input data bits are sent to all branch and leaf nodes. Multiple relay nodes could be used back-to-back to break critical path between geographically separated config nodes. Each config node is instantiated with a unique identifier and it only responds to the data designated for it. This allows a particular config node to be active without interfering other nodes even though all nodes receive the same inputs. The design therefore offers safer configuration than a traditional scan chain does where all nodes in the scan chain need to be configured at the same time.

2.1 Communication Protocol

Our communication packet shown in Table 1 for a config node is comprised of a *header* and *framed data*. Different fields in the header and frames in the data are separated by *framing bits*, which in our implementation is a single **0** bit. During communication period the *valid* field is pushed into a config node first, followed by the remainder of header and data. In our design the *valid* field is a two-bit vector "10" ('0' as the least significant bit) which indicates a valid packet has been received and that is the beginning of a valid packet. Next to the valid field is the *packet length* and its value equals the number of bits in one complete packet (header + data, including framing bits). The *node id* is an identifier to inform the recipient node to accept the packet. Following the header is the *framed data* composed by inserting framing bits every 8 bits in the data payload. The length of header is the same for all config nodes, while the framed data varies depending on the number of configurable pins connected to a particular config node.

f	framed data	*f*	node id	*f*	packet length	*f*	valid

Table 1: Communication Packet
Letter *f* denotes framing bits

Because there is no routing information in the communication protocol, configuration packets are broadcast to all config nodes in the network via the single serial communication line. Each config node checks the valid bits and node id in the header. If an arriving packet is valid and designated for itself, the config node will decode the framed data and apply the value to corresponding programmable pins; otherwise, the node simply ignores the amount of bits specified by the packet length in header.

Bit '1' in the valid field is inserted deliberately to make an all '0' packet not meaningful to any node; therefore we can use consecutive '0's as an NOP for all nodes. NOPs are useful to flush the configuration network without impact on any configured value, or to slow down the configuration data changing speed to avoid packet loss in case the configuration network is driven by a faster clock than the rest of the chip (refer to Section 3.2).

2.2 Config Node

Figure 2 shows the top-level diagram of a config node block. Ports and parameters are explained in Table 2 and Table 3 respectively.

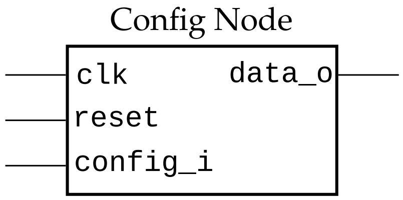

Figure 2: Config Node

Port	Direction	Note
clk	input	clock input for the destination clock domain
reset	input	reset input for the destination clock domain
config_i	input	encapsulated configuration input
data_o	output	configuration value to be connected to the programmable pins in the destination clock domain

Table 2: Config Node Ports

The encapsulated *config_i* input is a SystemVerilog packed struct shown below that is to ease signal assignments in Verilog coding.

```
typedef struct packed {
  logic cfg_clk;
  logic cfg_bit;
} config_s;
```

Parameter	Type	Note
id_p	decimal	an unique identifier for each config node
data_bits_p	decimal	number of bits in data payload equal to data_o width and the number of configurable pins
default_p	binary	safe default configuration value for pins

Table 3: Config Node Parameters

Apart from the parameters configurable when a config node is instantiated, there are several local parameters same for all config nodes, which are hidden but the

Local Parameter	Value	Note
valid bits	2	number of bits for the valid field in header
packet length bits	8	number of bits to hold a packet length value
node id bits	8	number of bits to hold a node id value
data frame bits	8	number of bits in a data frame
reset sequence bits	10	number of consecutive '1' bits to derive a configuration reset signal
synchronizer length	2	number of flip-flops in the clock domain crossing synchronizer

Table 4: Config Node Local Parameters

user must also keep them in mind when using a config node or composing a communication packet. Those local parameters are shown in Table 4. Therefore when assigning a node id the integer value should not exceed the maximum unsigned value that *node id bits* can represent. Similarly when composing a long packet its length value should fit in *packet length bits*.

Each config node contains a shift register used to buffer serial inputs from config_i and the shift register has the same structure depicted in Table 1. Because the data payload width varies between different config nodes, shift registers also have variable lengths to store framed data. Every config node monitors its shift register and once it detects a valid signal and a matching node id, it extracts payload from the framed data and assigns corresponding bits to data_o; if the packet is valid but the node id does not match, the config node loads the value of *(packet length - valid bits)* into a counter and does not respond to the shift register again (except reset, see Section 2.4) until the counter decreases to zero.

Once a new configuration value is obtained it is first stored in a register of the same clock domain as the shift register, and one cycle after the new value is registered the config node toggles a single bit *ready* signal to inform the destination clock domain. Because the destination clock and the configuration clock may differ, delay for one cycle here is to provide extra time for all the data register outputs to become stable before they are read in the destination clock domain.

By using the one-way ready signal's transition as a flag to transmit data crossing clock domain, we benefit from two ways. One is to ensure the value stored in the configuration clock domain registers is sampled exactly once when we believe it's stable, and the other is to eliminate the cost of a feedback acknowledge path, which by itself is another crossover signal and raises the chance of circuit going metastable (refer to Section 3.1).

2.3 Relay Node

A relay node block is illustrated in Figure 3.

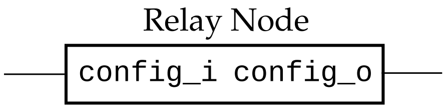

Figure 3: Relay Node

A relay node does not perform any processing on the data packets and just delays the inputs by one cycle and forwards them losslessly from input to output. Relay nodes are used to construct a configuration network and break critical path between geographically separated config nodes.

2.4 Resets

In the configuration clock domain, registers for counters and payload data need reset at the beginning of use. However, because we only have two spare I/O pads on the GreenDroid package, there is no way to use an additional signal from the outside of the chip and thus the reset must be generated from the serial inputs. To resolve this we use a sequence of bits from the shift register to derive a reset, and those bits are defined as the least significant *reset sequence bits* mentioned in Table 4. At any point in time, if a config node sees the reset sequence in the shift register, a reset signal is asserted and that is used to clear the counter and load *default_p* into the data payload register. Our reset sequence is defined as a vector of *reset sequence bits* long comprised of all '1's. The framing bits of '0's we inserted deliberately therefore are crucial to prevent a reset sequence from appearing in a valid configuration packet.

In the destination clock domain, a reset input is available from a port of a config node, which is supposed to be the same reset for the rest of GreenDroid chip. The reset for most of the GreenDroid chip is a synchronous level-sensitive signal, while in the destination domain of a config node the reset is specially designed to be synchronous but **edge-sensitive**. This distinction in reset is a failsafe design in case metastability (see Section 3.1) seriously impedes normal operating of GreenDroid. In this scenario we can assert the same reset for both config nodes and the rest of the chip, and perform pin configuration after the config node recovers and the rest circuit still remains in reset state.

3 Clock Domain Crossing

The configuration network is designed to work in a possibly different clock domain than the rest of the GreenDroid chip. The clock domain crossing (CDC) signals pose a challenging issue for design verification because traditional functional simulation using tools like VCS is inadequate to prove crossing signals also work properly. Within one clock domain, proper static timing analysis (STA) can guarantee that input data changes do not violate clock setup and hold time constraints of the sampling registers; when signals cross from one clock domain to another unrelated clock domain, STA is not enough to rule out metastability since CDC signals can change at any time with reference to the destination clock edges. Therefore to avoid functional errors that can only be detected in late design cycles, additional effort is taken to raise the level of confidence in our design.

3.1 Metastability

The proper operation of a clocked flip-flop depends on the input signals being stable within its required setup and hold time window. If the timing requirements are met, the correct output signal appears after a maximum delay of t_{co} (clock-to-output time). However, if the setup or hold time constraints are violated, the flip-flop output may take a much longer, or even infinite time to resolve to a stable state, and the state to which it resolves to is also non-deterministic. This unstable behavior of a flip-flop is referred to as **Metastability** as depicted in Figure 4. Metastability events may cause wrong signals being latched in the destination flip-flops and propagate through the rest of the circuit and therefore fail the proper function of the chip.

Designers often use the following synchronization techniques [3] to reduce the chance of metastability.

- A chain of flip-flops for 1-bit crossover signal
- Gray-encoding for vector crossover signals
- Handshaking synchronizers for control signals and MUXes for data
- Dual-clock FIFO

Variations of these common techniques are used in different design scenarios. In our work, we use a combination of a flip-flip sequence and a data MUX, on top

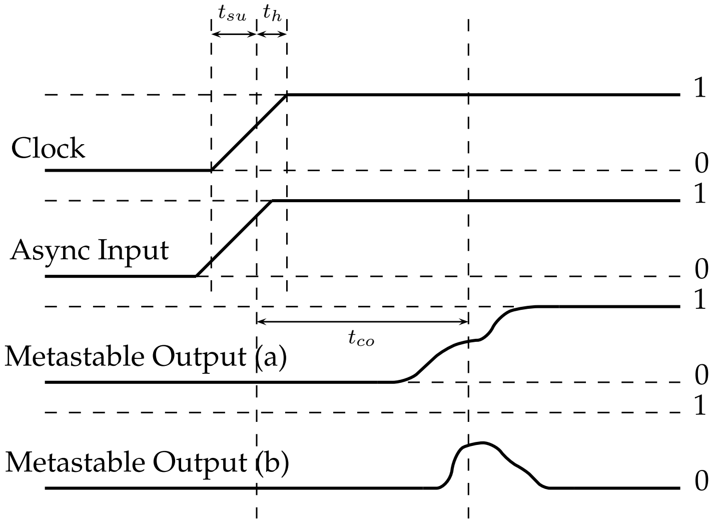

Figure 4: The Metastability Behavior

of that we develop a customized synchronizing protocol for the control path and clock domain crossing stabilizers along the data path. The RTL diagram of our design is depicted in Figure 5.

3.2 Control Path Synchronizer

As described in Section 2, the source clock domain uses a single bit alternate *ready* signal to indicate that a new configuration value is ready and stable to be sampled. In the destination clock domain, two flip-flops are XOR-ed to detect the edge of a transitioning ready signal. A parameterizable-length (at least two) sequence of flip-flops is inserted between the source ready and the first edge-detecting flip-flop to provide extra amount of time for the possible metastable value to resolve and lowers the probability that the design fails.

Successful operation of our design is based on the assumption that data in the source clock domain registers does not change before it gets sampled in the destination domain. Since there is no feedback loop in our design, we guarantee this by feeding the configuration network with properly designed input sequence to ensure data is stable during the communication period, which is the length of the synchronizer chain plus three flip-flops (one delay flip-flop in the source clock domain and the two edge-detecting flip-flops in the destination).

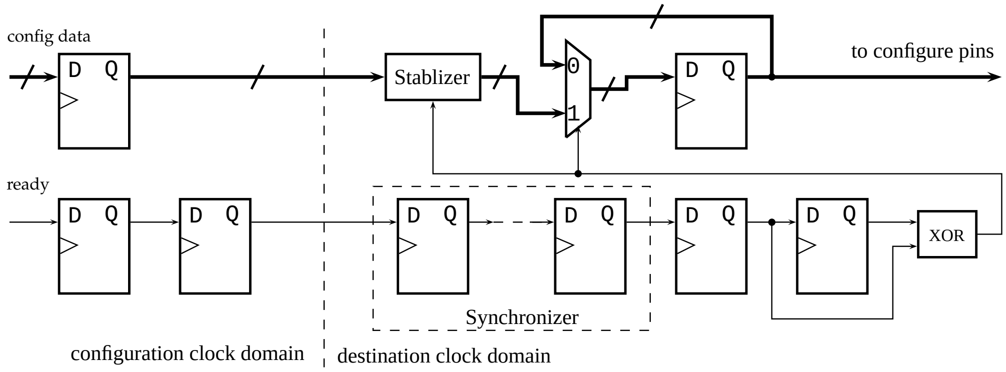

Figure 5: Our Clock-Domain Crossing Circuit

Compared to a full handshaking communication, the one-way control signal design has the following advantages:

- Save logic to implement a feedback path.
- Reduce the chance of circuit metastability because the feedback path also crosses clock domains.
- Respond faster and deliver higher throughputs in transferring data.

and the only disadvantage is that it cannot stall the source data pipeline and may lose packets if data from source clock domain is changing too frequent. In our design, we are able to lower the configuration data changing frequency by inserting NOPs from the input of the network; and therefore eliminate the chance of data loss, while taking advantage of the benefits from the one-way crossing.

Although the use of synchronizers can not completely eliminate the chance of entering metastable state, the probabilistic function of Mean Time Between Failure (MTBF, refer to Section 3.4) qualifies the circuit to be continuously operating in acceptable duration of time. In RTL development stage of a chip, users of this design can increase the MTBF by extending the length of the synchronizer chain.

3.3 Data Path Multiplexer

The control signal is typically considered to be safe by using the aforementioned synchronizers, and our design makes sure the asynchronous data path is through when a transition of ready is detected, i.e., when the *select* input to the data MUX is logic 1. Nevertheless, one commonly overlooked potential cause of metastability is the use of this asynchronous data MUX when its select input is logic 0. Ideally, the MUX should shut off the asynchronous data path completely when its select input is logic 0, but since the physical implementations of MUXes vary between different process design kits, we are not convinced that the MUX always works as we expected. During normal operation of the circuit, toggling of ready does not happen frequently and this means the destination register at the crossing boundary spends most of its time pulling data from the MUX's data path selected by logic 0. Although this data path is synchronous with respect to the destination clock, we need extra measures to make sure the stability of this data path is not interfered by the asynchronous data input. Interferences may also cause unstable signals reach the register's input within its setup-hold time window. Therefore we introduce the *clock domain crossing stabilizers* right before the MUX's asynchronous data input to provide extra protection for the synchronous logic.

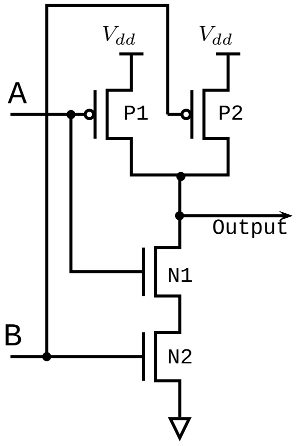

Figure 6: 6-MOSFET NAND2 Gate

Our stabilizer is implemented by exactly one NAND2 gate, whose behavior is predictable, and implementation is simple enough to be done in typical six MOSFETs, as shown in Figure 6. When one input, for example B, to this NAND2 gate is stable logic 0, obviously N-MOS N2 is shut off and no matter how input A varies, the NAND2 output is strongly charged by V_{dd} and immune to changes of input A. By contrast, if B is stable logic 1, then N-MOS N2 is on, and now if input A voltage varies, the NAND2 output also changes accordingly until it resolves to a state determined by input A. Corresponding voltage divider models are presented in Figure 7 and Figure 8, respectively. Same analysis also works if we choose A as the stable input.

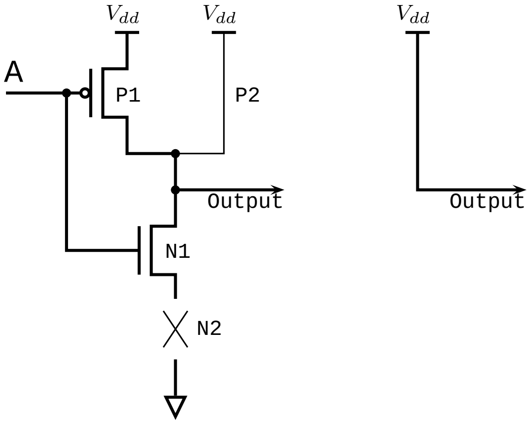

Figure 7: NAND2 Gate with a Stable Logic 0 Input

Deliberately we connect one input of the stabilizer to the same select signal of the data MUX, and the other input to one asynchronous data line. By this means, we believe the asynchronous data fluctuations are well filtered by the stabilizers, and passed to the destination registers only when the crossover data is stable, which is signaled by a logic 1 on the select line.

Noticeably valid configuration signals are inverted after NAND2 gates, and recovery inverters are inserted after the sampling registers. At the cost of a few more MOSFETs, we raise our confidence in this clock domain crossing design. Use of such stabilizers is **not** merely bound to our circuit and can also be applied to other clock domain crossing cases whenever crossover data needs to be blocked.

3.4 Mean Time Between Failure

Although the use of synchronizers can reduce the chance that metastable signals cause the circuit fail, the chance cannot be completely avoided in a clock domain crossing design. Typically the metric of MTBF is employed to evaluate the likelihood of a circuit fail.

Entering a metastable state is a probabilistic function with reference to the destination clock frequency, the asynchronous signal transitioning frequency and constants that defines the window in which a transition can cause metastability. A typical formula [1] [3] [4] [5] to calculate the MTBF of a two-flip-flop synchronizer sampling asynchronous data is given in (Equation 1).

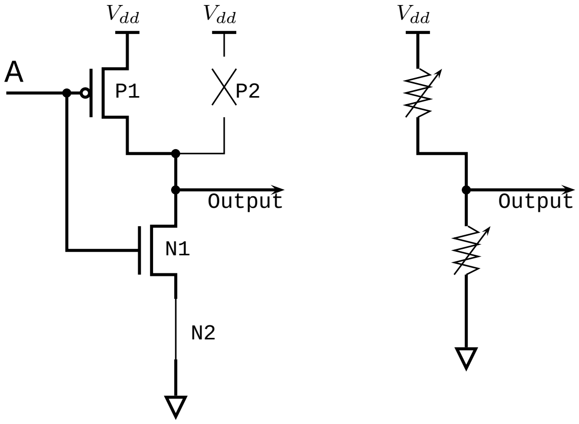

Figure 8: NAND2 Gate with a Stable Logic 1 Input

$$MTBF = \frac{e^{\frac{T_r}{C_2}}}{C_1 \times f_{dst} \times f_{data}} \tag{1}$$

In (Equation 1), C_1 and C_2 are constants that depend on the tape-out process and the operating conditions of the flip-flops; T_r is the maximum allowed amount of time for the first flip-flop to resolve a metastable state; and f_{dst} and f_{data} are the frequencies of the destination clock and the asynchronous data inpu7(n)-0.695,a re(p)-27.4505(e)0.0490

4 Conclusions and Future Work

This technical paper has presented the configuration network design for our GreenDroid chip. We believe this design is better and safer than a traditional scan chain for the same purpose. We have also taken strong measures in our design trying to reduce the chance of metastability caused by clock domain crossing. The design will be evaluated by FPGA simulation and eventually validated by testing the GreenDroid chip. At this point in time, this configuration network is write-only, which means data can only be pushed into configurable targets. Adding support for reading output status of IPs is also useful for system status monitoring.

References

[1] Altera. Understanding Metastability in FPGAs. *Altera White Paper*, 2009.

[2] M. Baghini and M. Desai. Impact of technology scaling on metastability performance of CMOS synchronizing latches. In *Design Automation Conference*, pages 317–322. IEEE, 2002.

[3] Cadence. Clock domain crossing - Closing the loop on clock domain function implementation problems. *Cadence Technical Paper*, 2004.

[4] D. Chen, D. Singh, J. Chromczak, D. Lewis, R. Fung, D. Neto, and V. Betz. A comprehensive approach to modeling, characterizing and optimizing for metastability in FPGAs. *Proceedings of the 18th annual ACM/SIGDA international symposium on FPGA*, 2010.

[5] C. Dike and E. Burton. Miller and noise effects in a synchronizing flip-flop. *IEEE Journal of Solid-State Circuits*, pages 849–855, 1999.