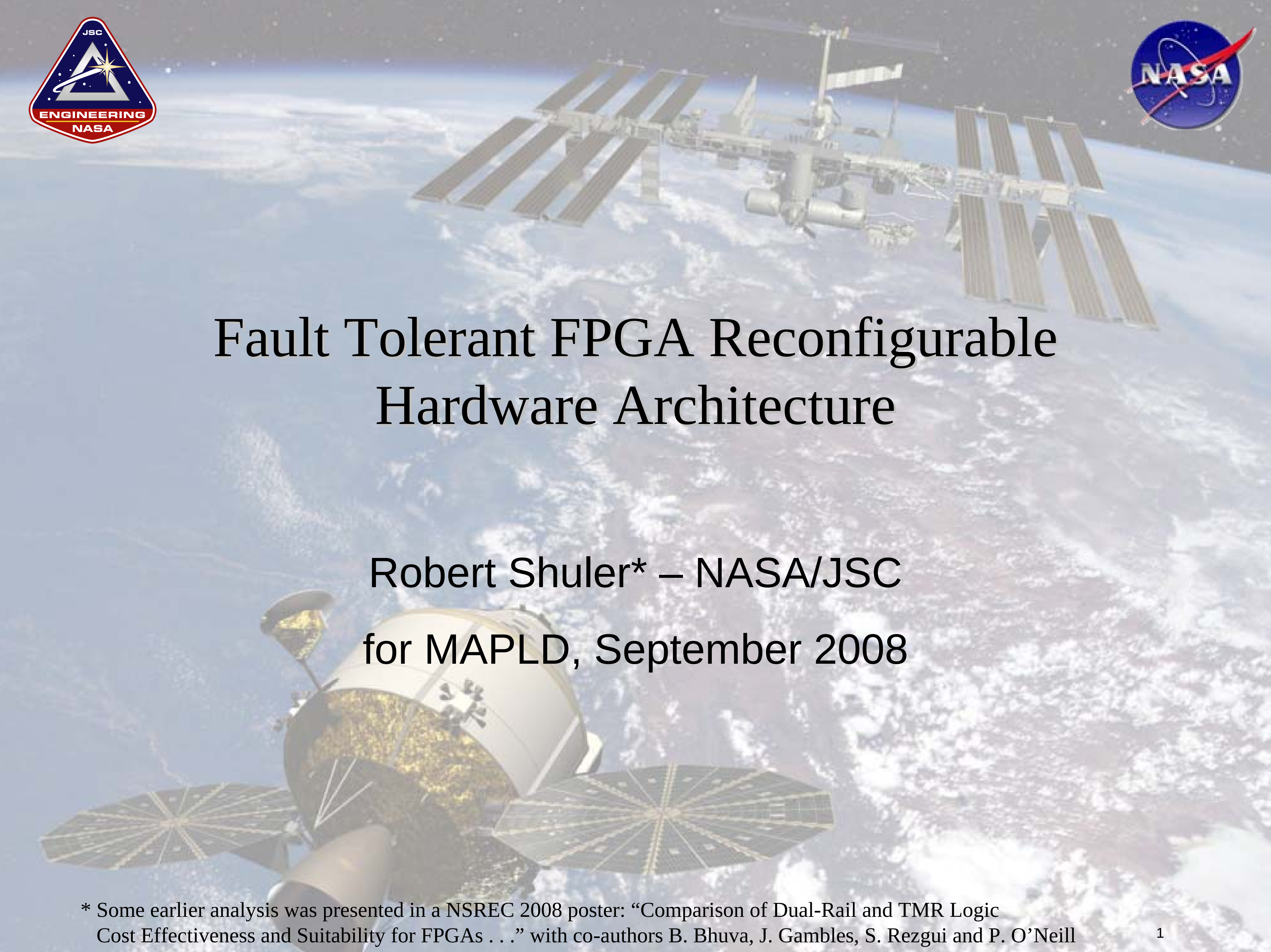


Fault Tolerant FPGA Reconfigurable Hardware Architecture

Robert Shuler* – NASA/JSC

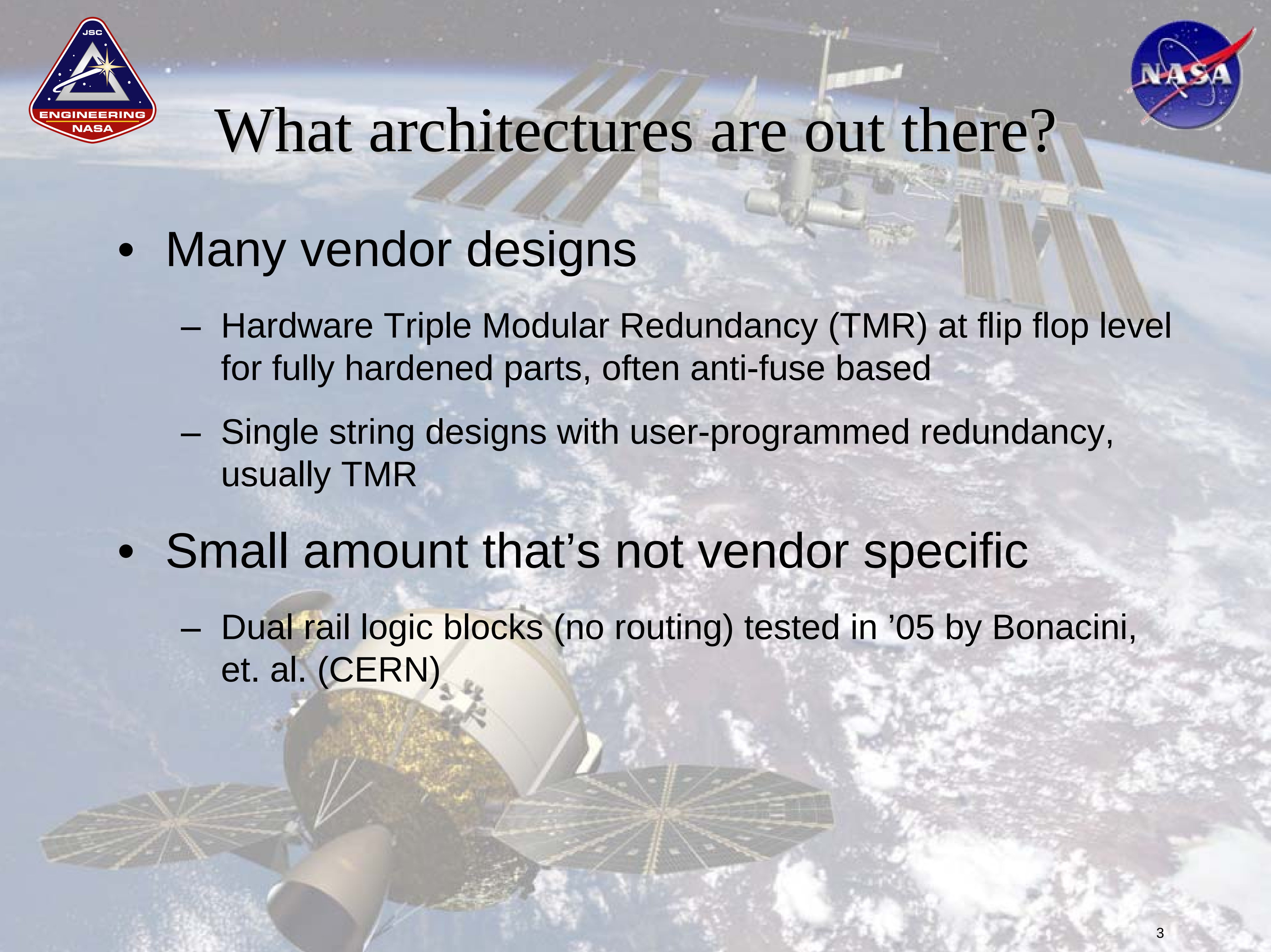
for MAPLD, September 2008

* Some earlier analysis was presented in a NSREC 2008 poster: “Comparison of Dual-Rail and TMR Logic Cost Effectiveness and Suitability for FPGAs . . .” with co-authors B. Bhuva, J. Gambles, S. Rezgui and P. O’Neill

Wanted to do three things…

- Apply several years of work in SEU/SET mitigation techniques
- Satisfy a sponsor’s interest in improving FPGAs for use in deep space projects
- Encourage researchers to investigate FPGA specific circuit topologies, e.g. LUTs, muxes and routing

Idea: Develop an FPGA architecture for R&D purposes



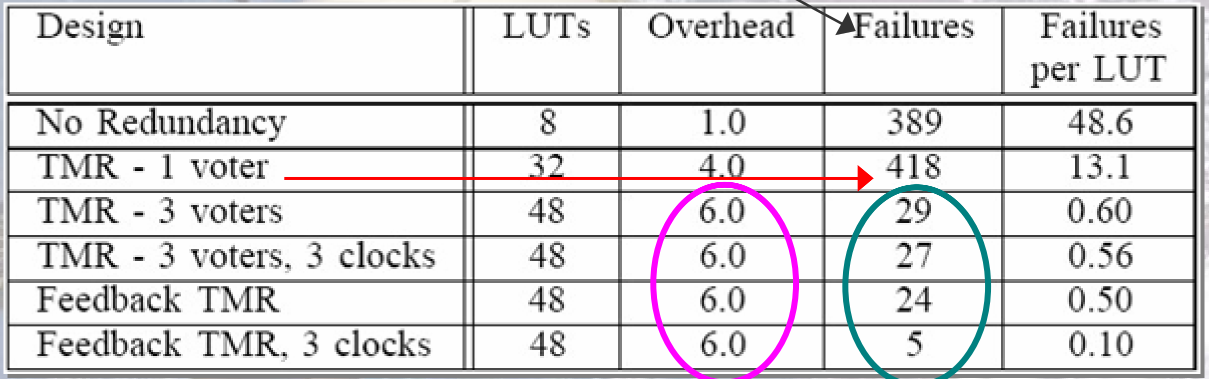
What architectures are out there?

- Many vendor designs
  - Hardware Triple Modular Redundancy (TMR) at flip flop level for fully hardened parts, often anti-fuse based
  - Single string designs with user-programmed redundancy, usually TMR
- Small amount that's not vendor specific
  - Dual rail logic blocks (no routing) tested in '05 by Bonacini, et. al. (CERN)

What’s out there? (cont’d)

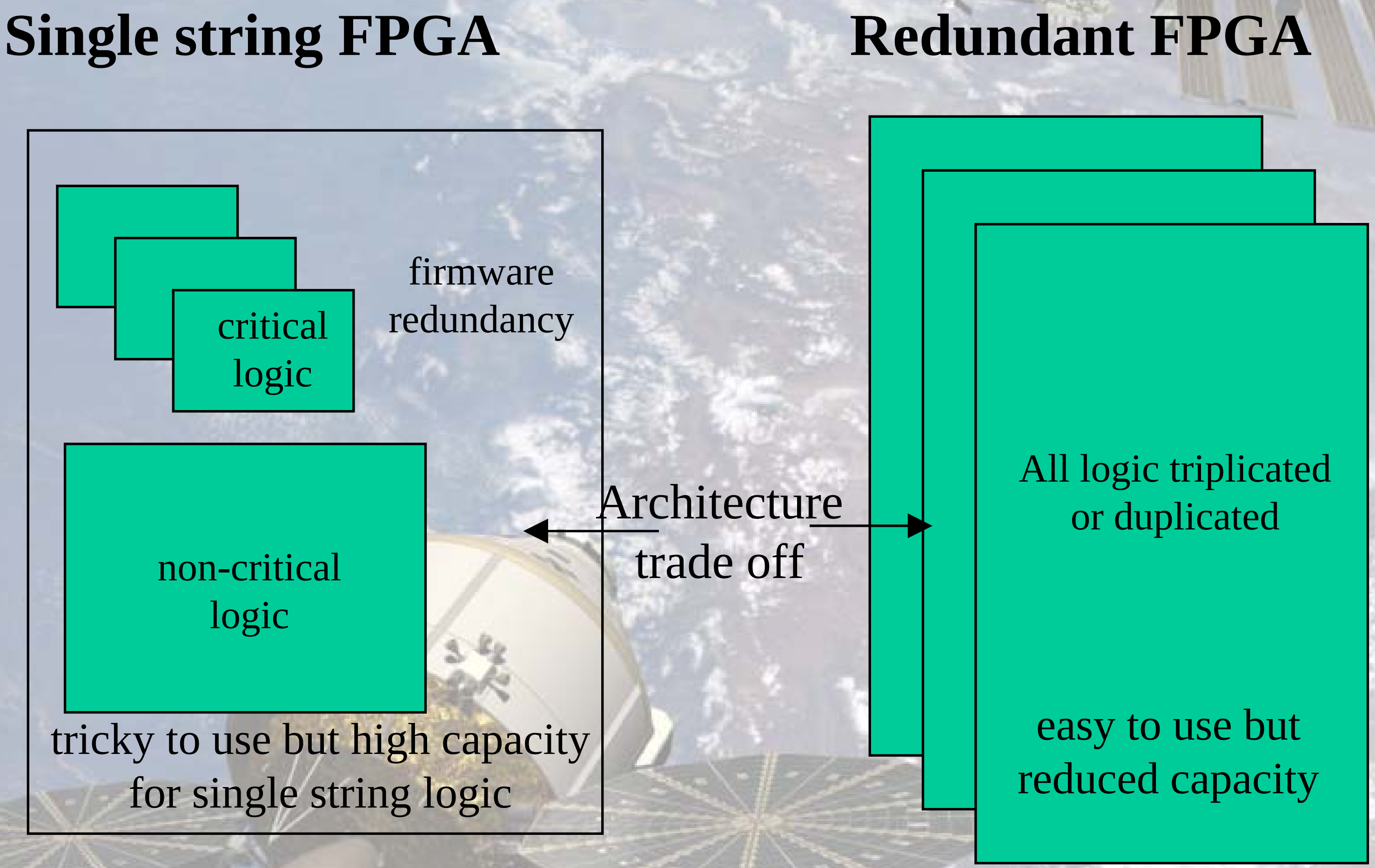
- Lots of interest in re-configurable FPGA
  - Scrubbing, e.g. internal vs. external, “one chip” TMR viability
  - Domain crossing errors (Quinn et. al. 2007)
  - Voting frequency (Pratt, Wirthlin, Quinn, et. al.)
  - TMR correctness (apparently there are many surprises)
  - TMR efficiency (Wirthlin et. al. 2003, table below)

sensitive configuration bits



Design	LUTs	Overhead	Failures	Failures per LUT
No Redundancy	8	1.0	389	48.6
TMR - 1 voter	32	4.0	418	13.1
TMR - 3 voters	48	6.0	29	0.60
TMR - 3 voters, 3 clocks	48	6.0	27	0.56
Feedback TMR	48	6.0	24	0.50
Feedback TMR, 3 clocks	48	6.0	5	0.10

Which approach to take?

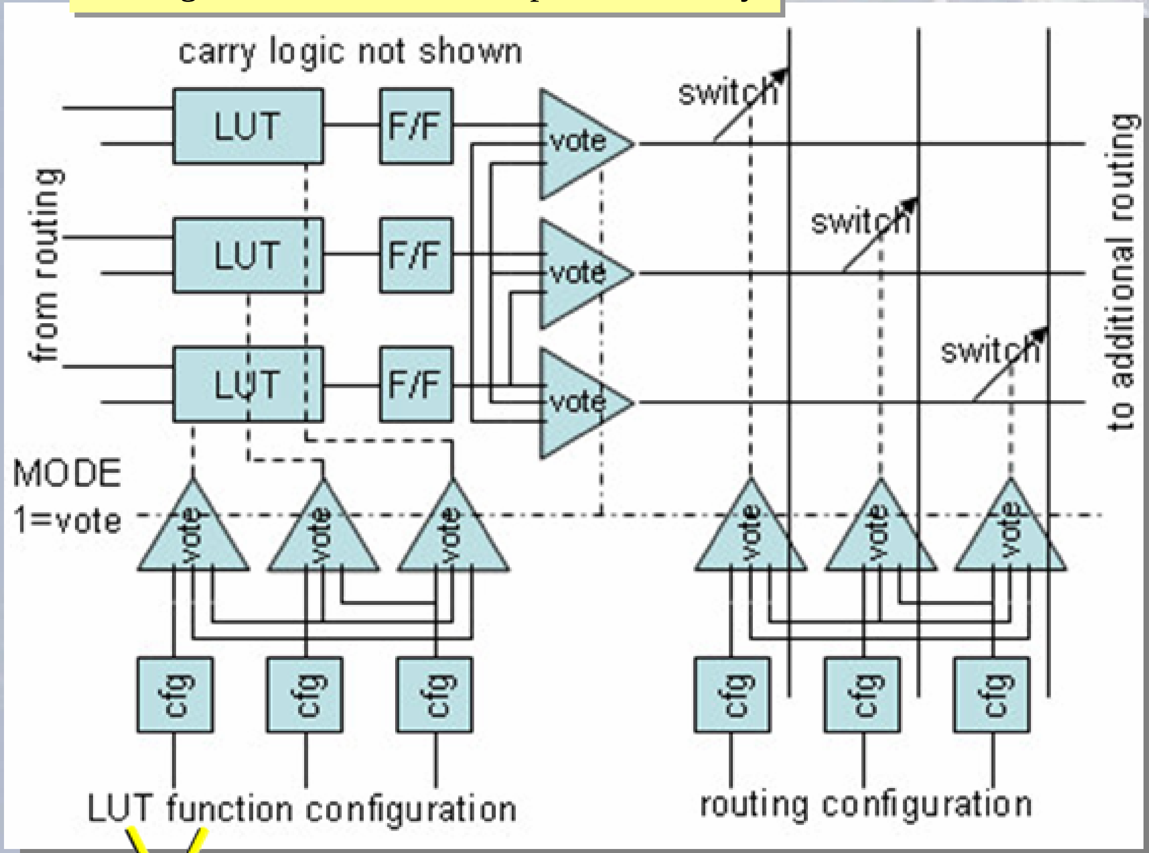


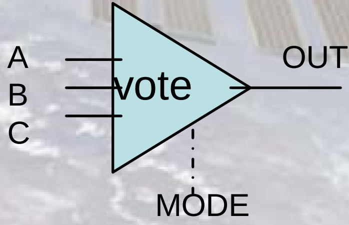
Single string FPGA
Redundant FPGA
critical logic
firmware redundancy
non-critical logic
tricky to use but high capacity for single string logic
Architecture trade off
All logic triplicated or duplicated
easy to use but reduced capacity

Capacity is a big driver for signal processing, robotics, many apps

Are we stuck with that choice?

One simplified FPGA logic block and one routing switch shown with triple redundancy



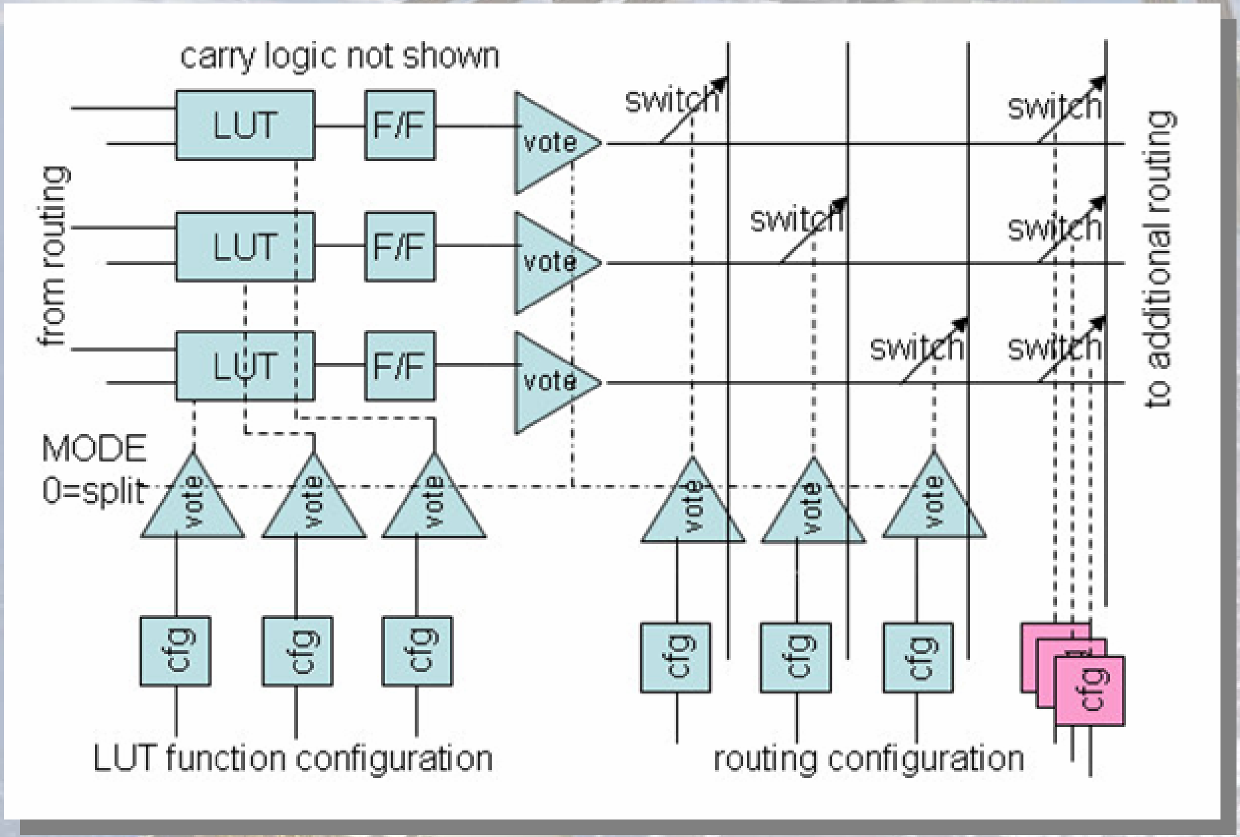


VOTE / SPLIT

MODE	A	B	C	|	OUTPUT
0	a	b	c	|	a
1	a	b	c	|	majority (a,b,c)

What if the TMR resources could be "split" in 3, each with its own programming resources?

What does it look like when split?



Three independently programmable domains are created

Need upper level routing hierarchy to communicate between domains (ordinarily present for larger capacity, not additional)

How much single string capacity can be retained? (efficiency)

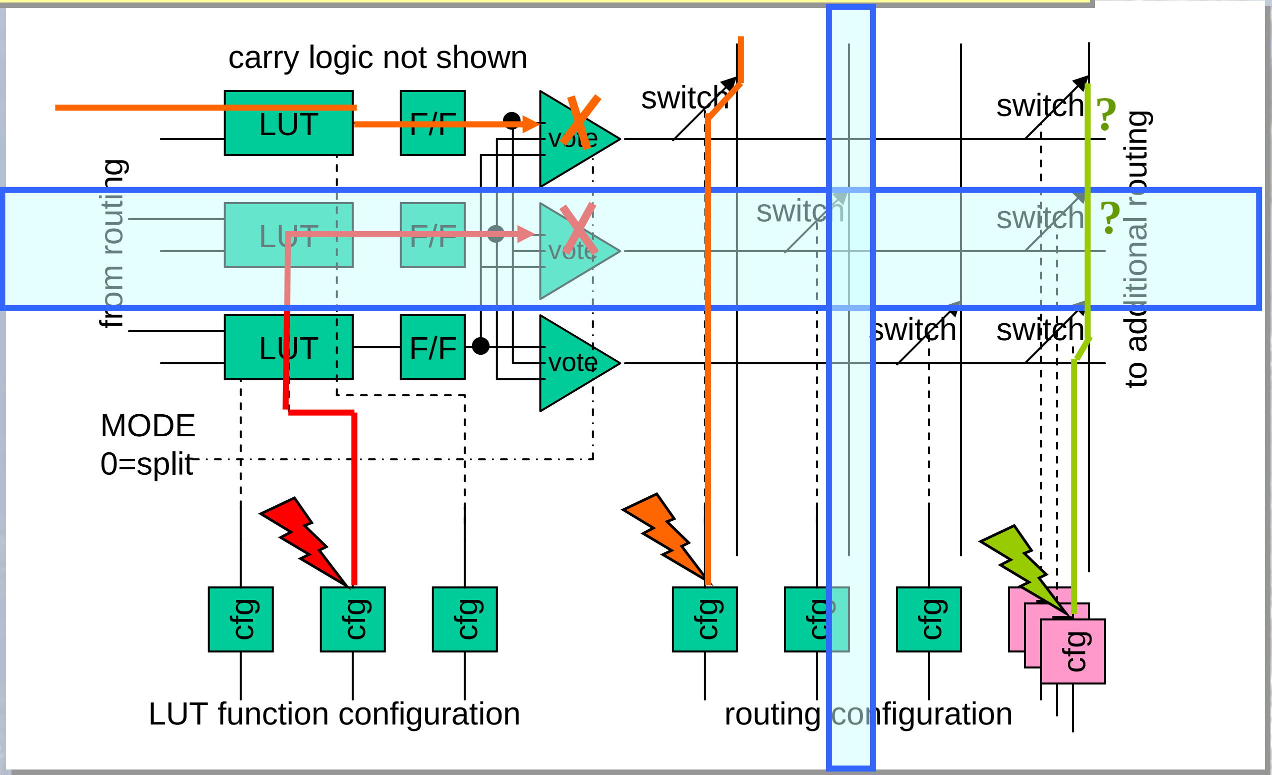
* type of SEU mitigation	redun-dancy	logic ratio	logic eff.	config ratio	config eff.	config portion	total efficiency
single string	1	1	100%	1	100%	75%	100%
TMR everything	3	3.8	79%	2.55	39%	75%	49%
TMR + trusted cfg.	3	3.8	79%	1	100%	75%	95%

- "single string" – an ordinary FPGA with no SEU/SET mitigation
- "TMR everything" – everything is triplicated and outputs are voted
  - User registers are always clocked so errors are corrected (synchronous design)
  - Configuration memory still needs mitigation (e.g. scrubbing)
- "TMR + trusted cfg." – configuration is not voted … at the time of this table it was thought it had to be flash or hardened SRAM

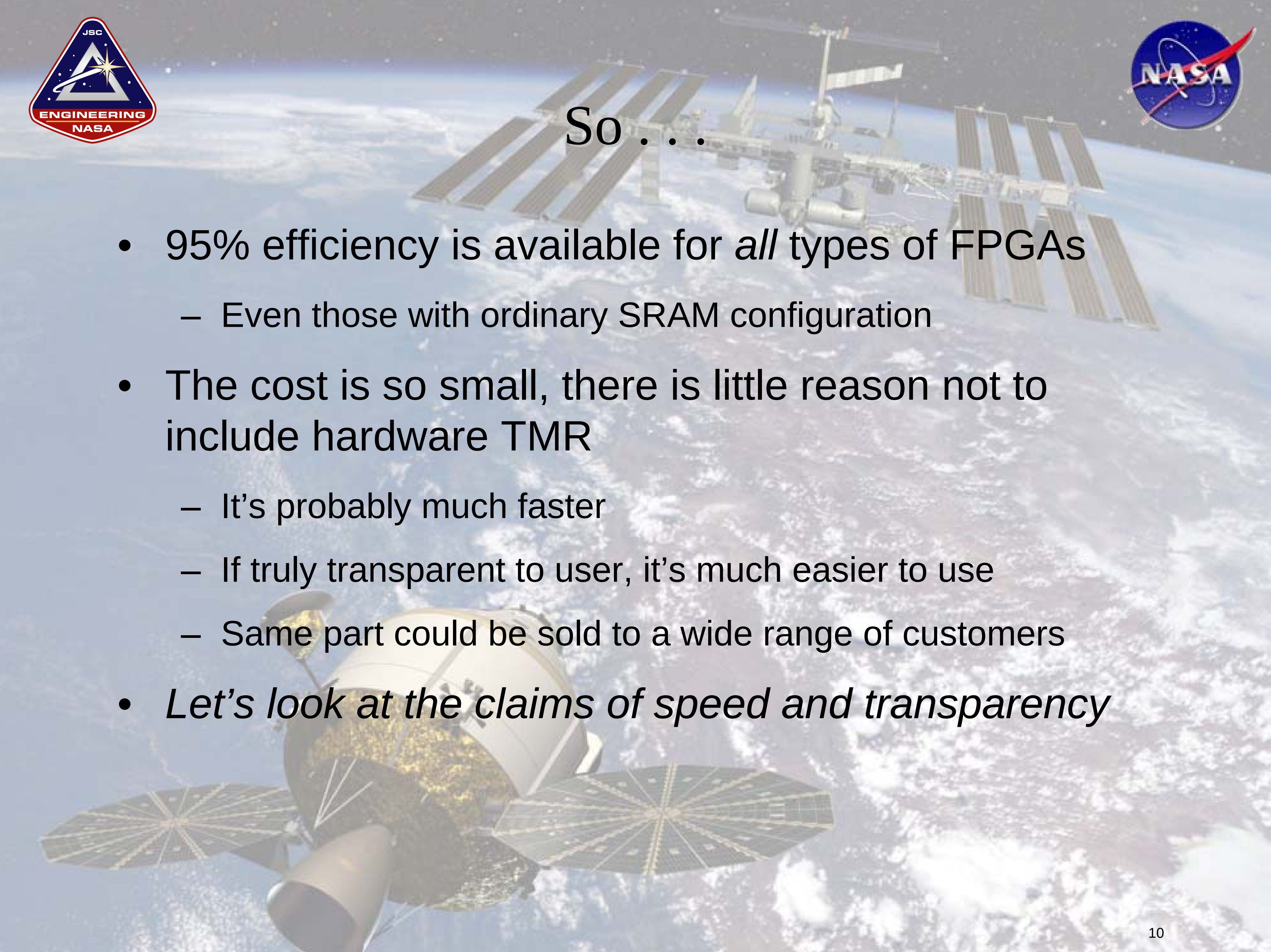
* This table is a subset of data presented in a NSREC 2008 paper: "Comparison of Dual-Rail and TMR Logic Cost . . ." with Bhuva, Gambles & Rezgui. The configuration memory portion of ~75% is from Morgan, et. al. IEEE TNS Dec 2007.

A surprising discovery…

TMR Splittable Architecture with un-voted configuration memory



- No single configuration bit can affect more than one voting domain, so configuration bit errors are voted out by logic/register voting
- Cross domain routing requires 2 errors (2 switches) to affect multiple domains

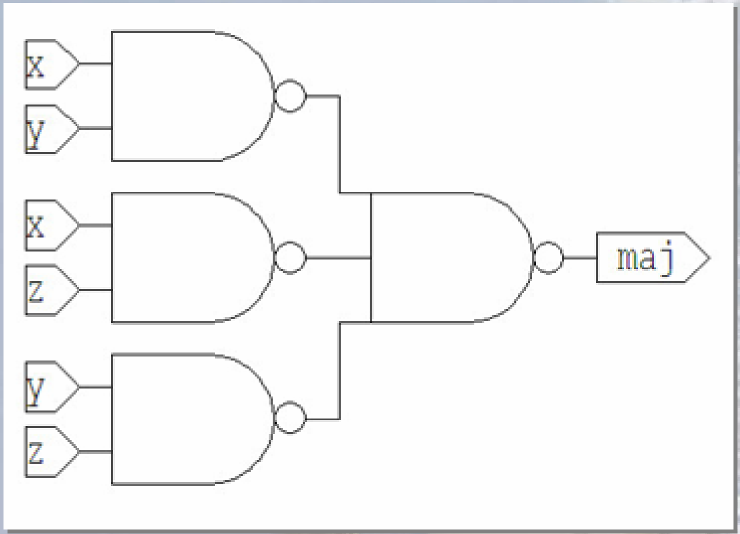


So . . .

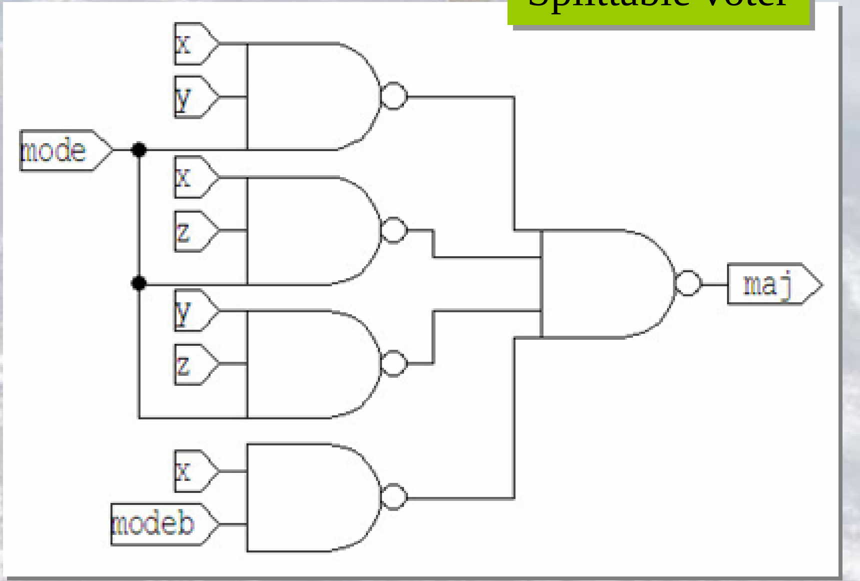
- 95% efficiency is available for *all* types of FPGAs
  - Even those with ordinary SRAM configuration
- The cost is so small, there is little reason not to include hardware TMR
  - It’s probably much faster
  - If truly transparent to user, it’s much easier to use
  - Same part could be sold to a wide range of customers
- *Let’s look at the claims of speed and transparency*

Voting scheme determines both speed and transparency

Normal majority voter



Splittable voter



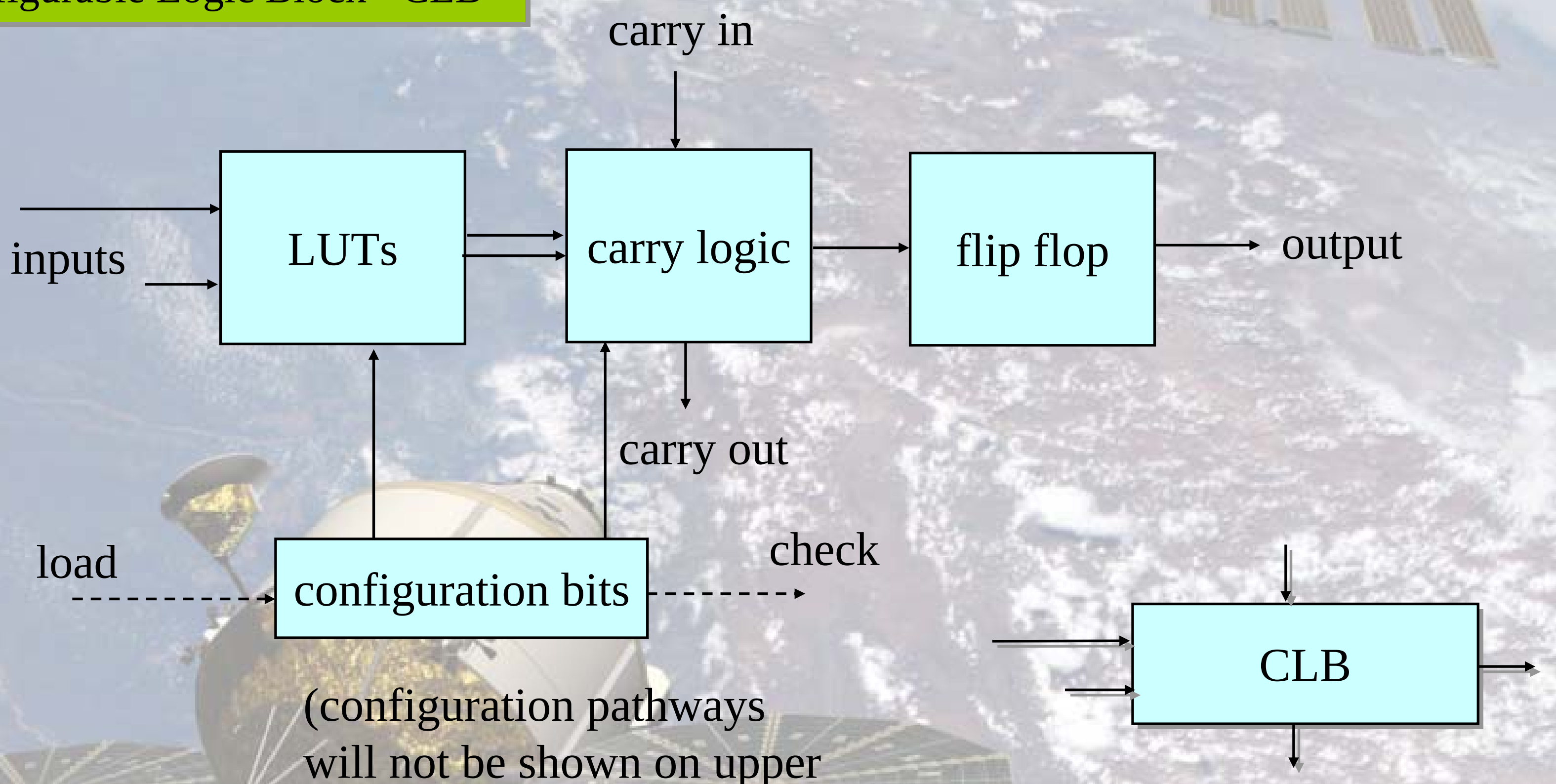
- Normal voter used in recent tests demonstrated <<10^{-11} errors/bit day
- Adds only 2 gate delays (vs. at least 1 routing and LUT delay for firmware)
- LUT and routing delay are usually an order of magnitude greater than gate delays! Saves at least one programmed “logic level”
- *Transparency will be shown in a demonstration application that follows . . .*

Demonstration Application

- FPGA with 6 CLB’s when single string, 2 when TMR
- Demo applications:
  - 6 bit counter for single string
  - 2 bit counter for TMR
- I/O routing not shown
- Redundant clock and mode lines not shown
- Goal is to illustrate routing, verify fault isolation, and demonstrate “transparency” of the redundancy

Building blocks…

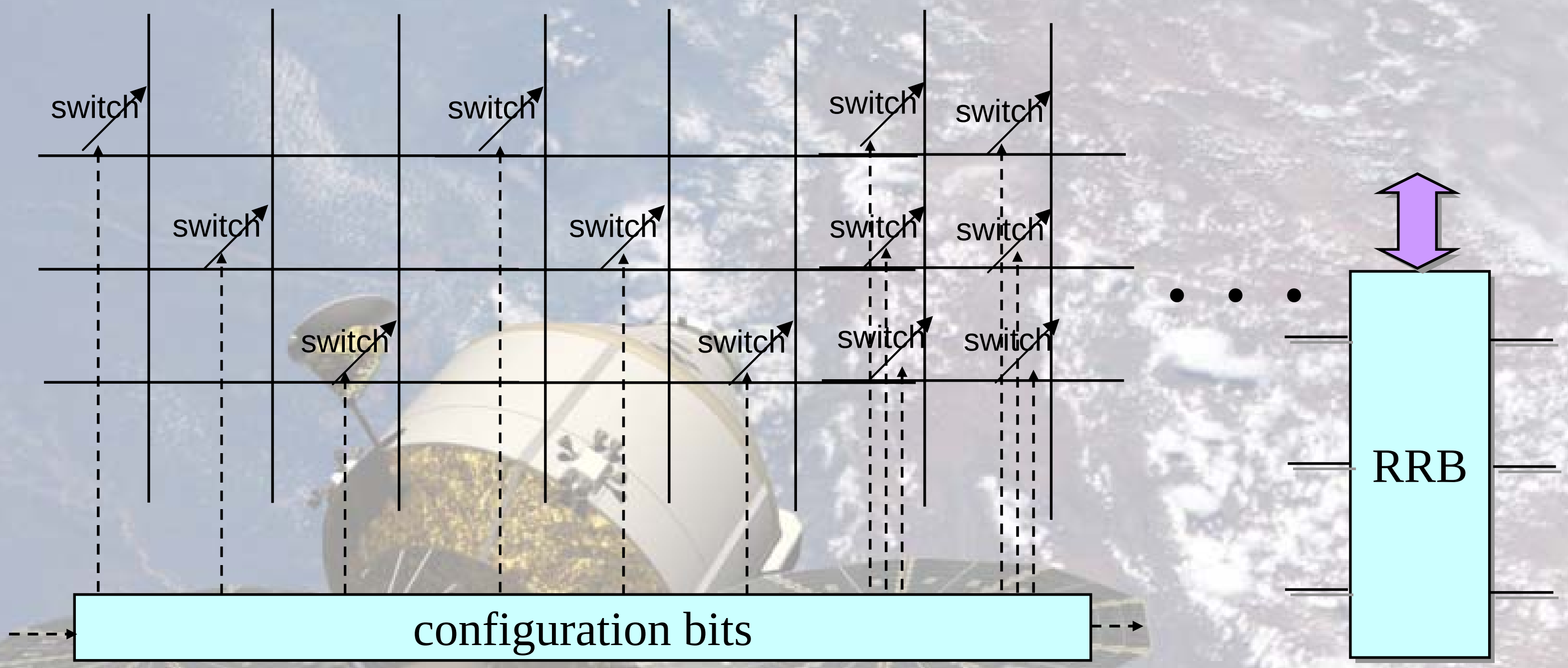
Configurable Logic Block - CLB



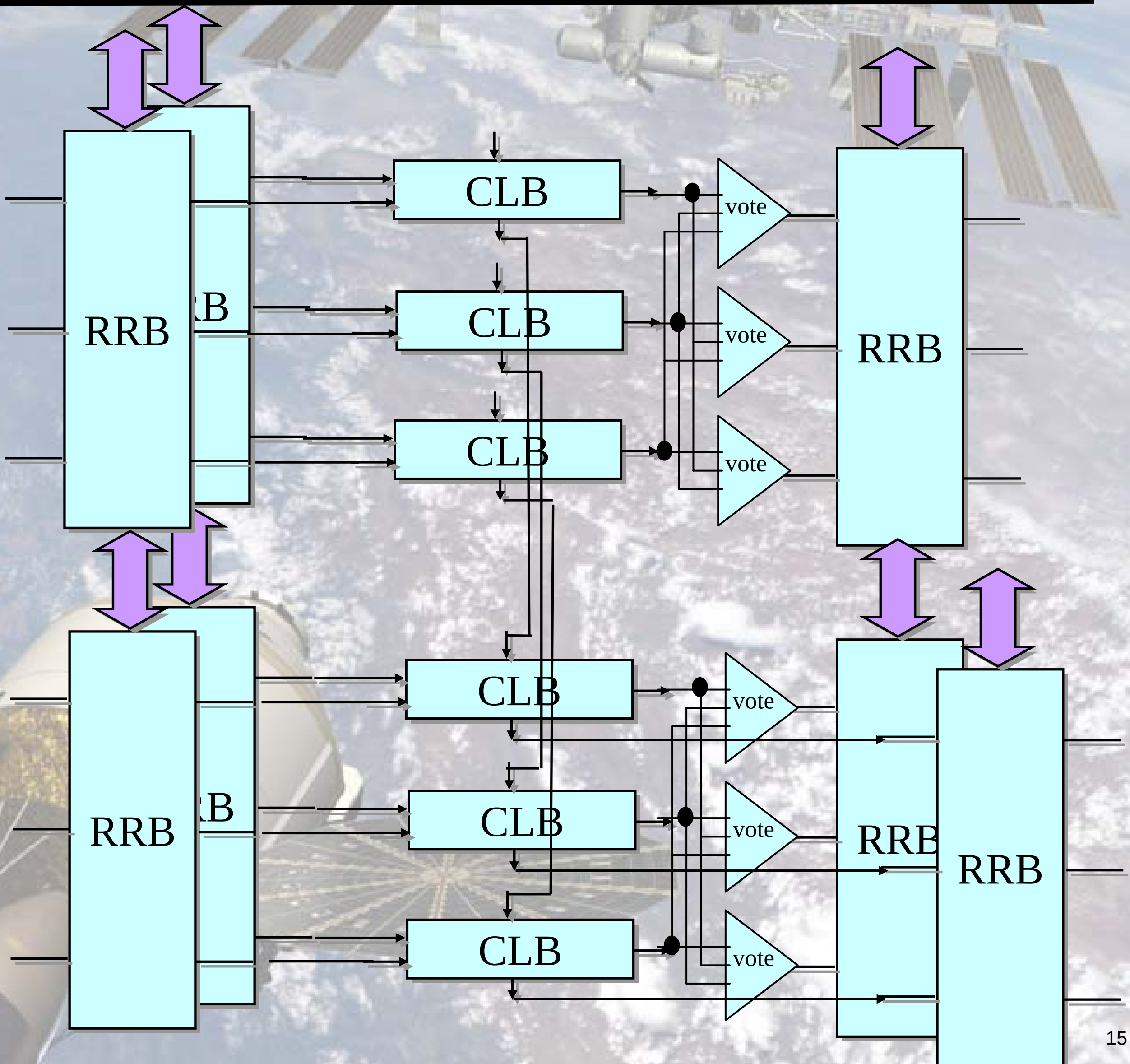
(configuration pathways will not be shown on upper level diagrams)

Building blocks…

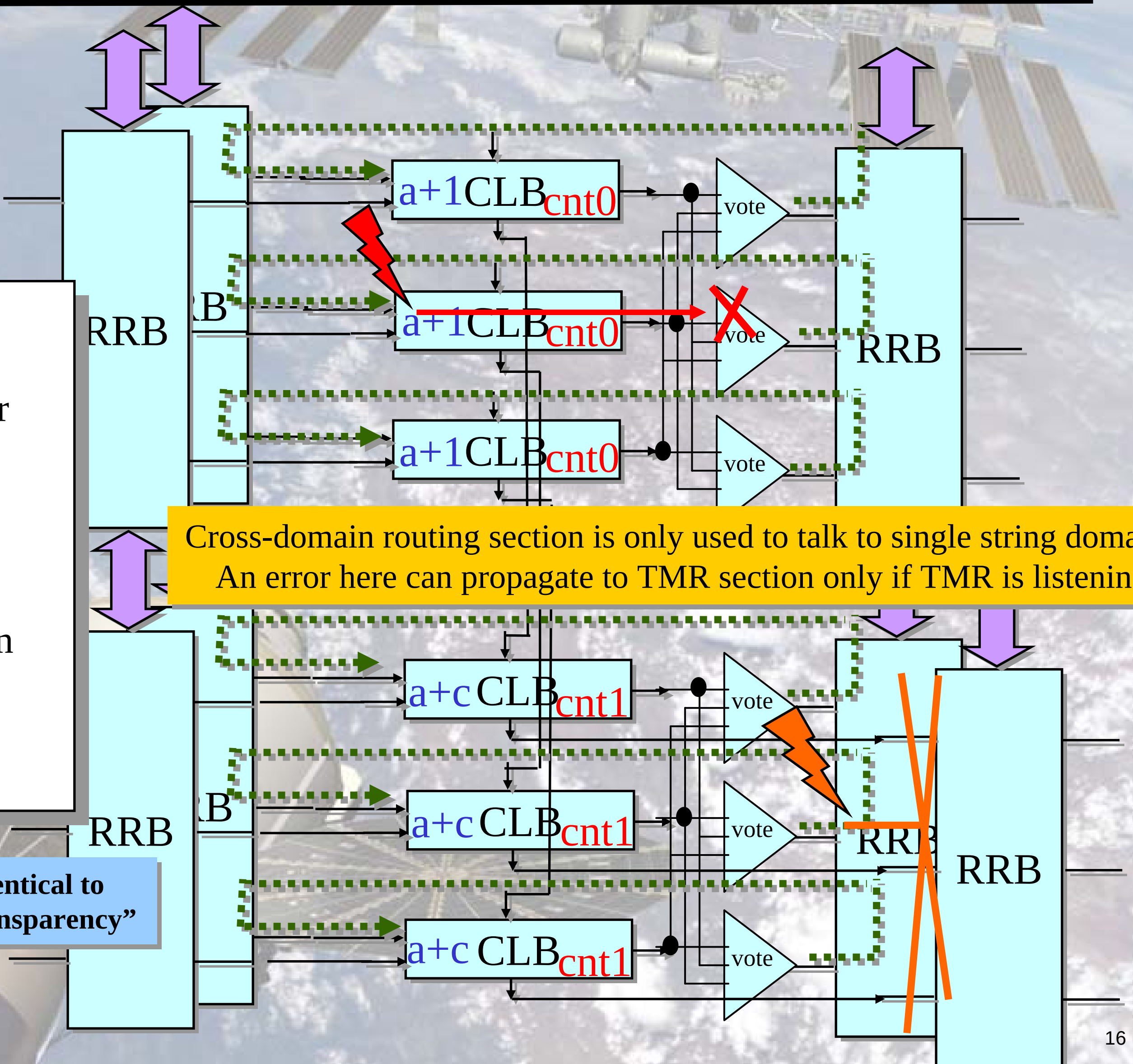
Redundant Routing Block - RRB



Un-programmed 6 CLB FPGA



2 bit TMR counter place & route

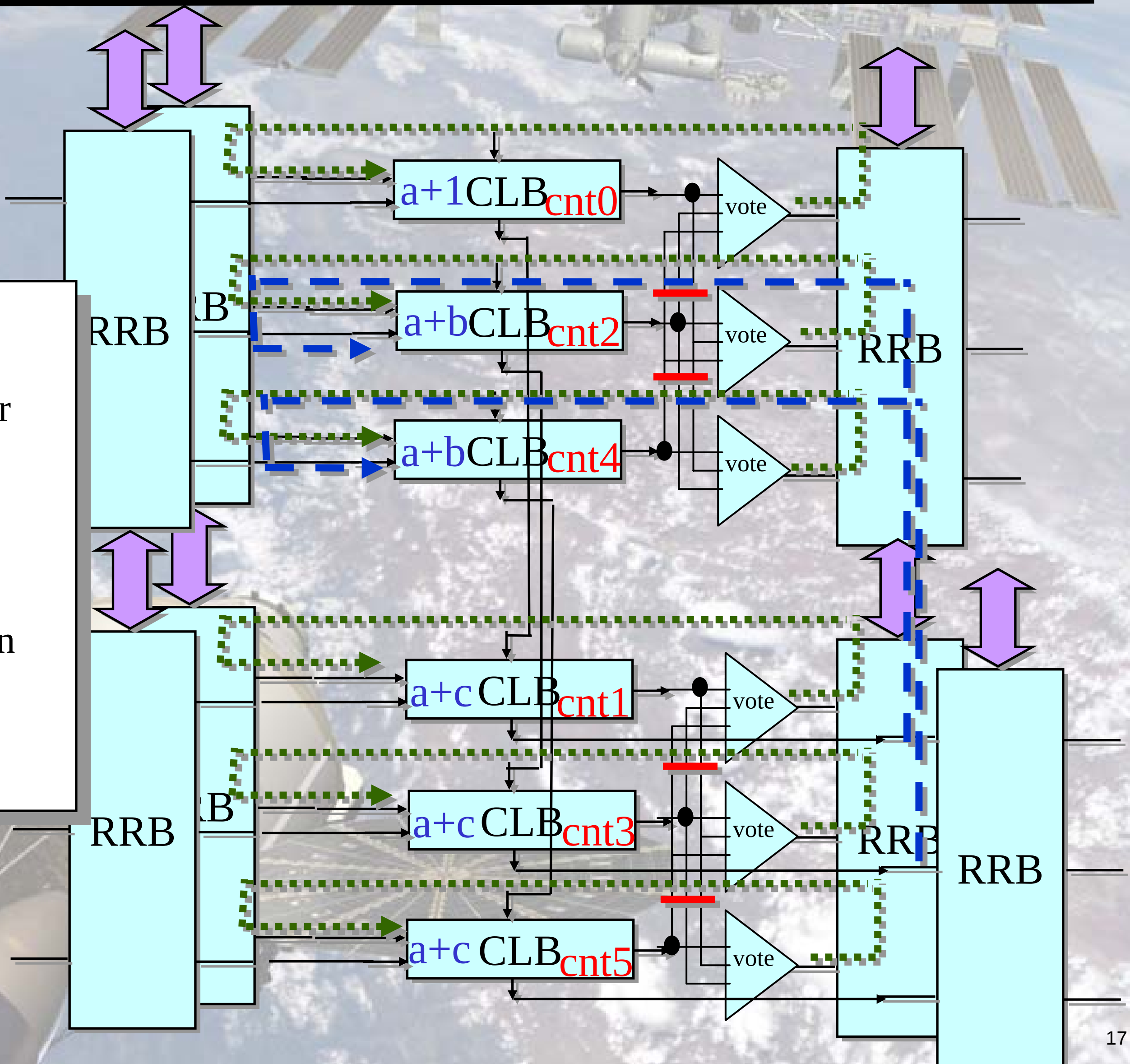


```
-- VHDL

signal a:std_logic_vector
   (1 downto 0);

begin
process (clk) begin
   if rising_edge(clk) then
      a <= a + 1;
   end if;
end process;
```

Cross-domain routing section is only used to talk to single string domains. An error here can propagate to TMR section only if TMR is listening.

programmed function is identical to original specification, i.e. "transparency"

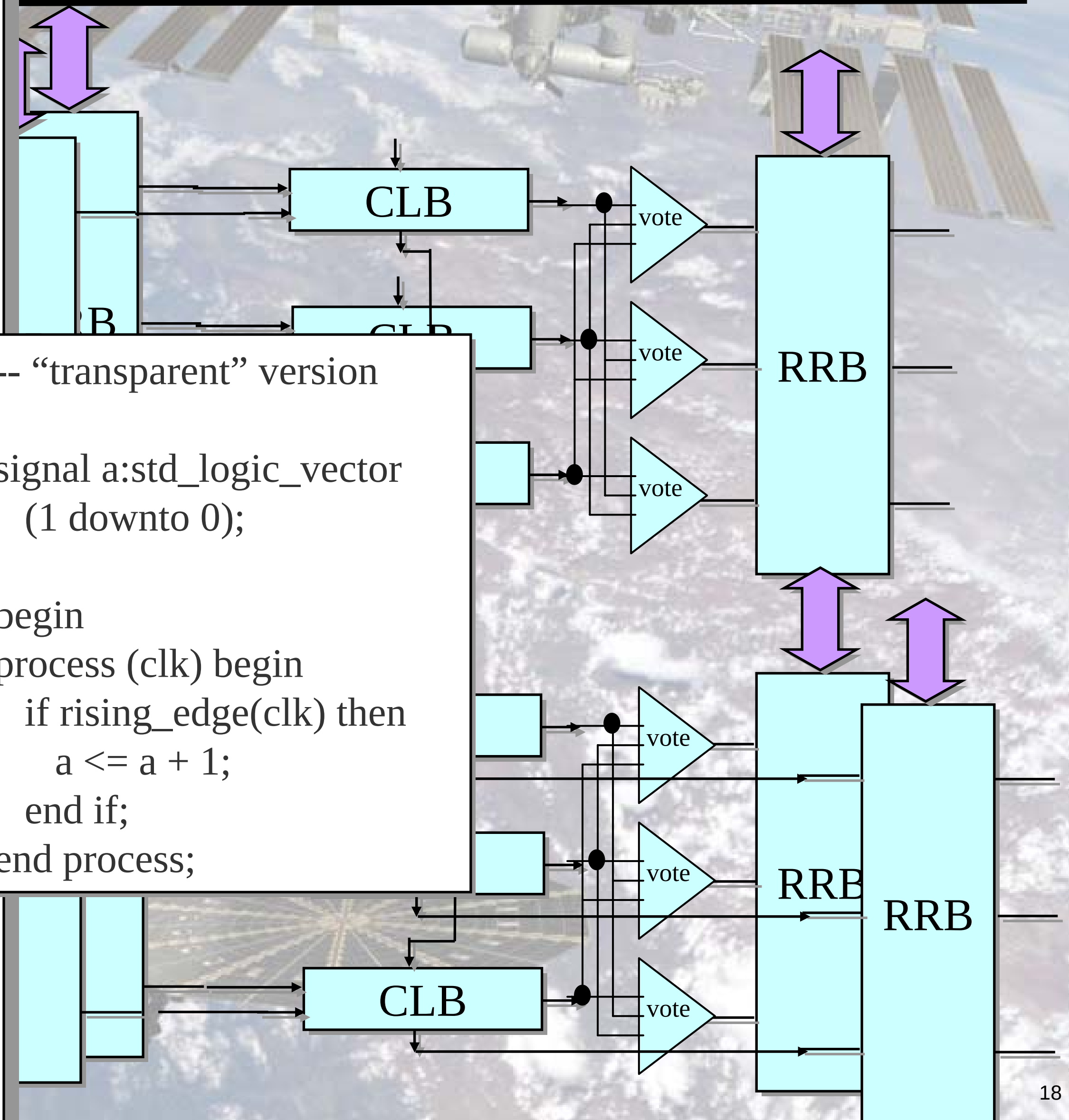
6 bit single string counter place & route



```
-- VHDL

signal a:std_logic_vector
   (5 downto 0);

begin
process (clk) begin
   if rising_edge(clk) then
      a <= a + 1;
   end if;
end process;
```

2 bit user defined TMR

```
-- user defined TMR
signal ax:std_logic_vector (1 downto 0);
signal ay:std_logic_vector (1 downto 0);
signal az:std_logic_vector (1 downto 0);

function majority (x,y,x:std_logic) is begin
  return (x and y) or (y and z) or (x and z);
end;

begin

process (clkx) begin
  if rising_edge(clkx) then
    ax <= majority(ax,ay,az) + 1;
  end if;
end process;

begin process (clky) begin
  if rising_edge(clky) then
    ay <= majority(ax,ay,az) + 1;
  end if;
end process;

begin process (clkz) begin
  if rising_edge(clkz) then
    az <= majority(ax,ay,az) + 1;
  end if;
end process;

--NOTE:  Systhesis optimization must be
--  suppressed to actually get redundancy!
```



```
-- “transparent” version

signal a:std_logic_vector
  (1 downto 0);

begin
process (clk) begin
  if rising_edge(clk) then
    a <= a + 1;
  end if;
end process;
```

What can we take away?

- Reconfigurable hardware fault tolerance in FPGAs is…
  - Low cost (5% of single string capacity, exactly 3x overhead for TMR)
  - Fast (saves at least one “logic level”)
  - Easy to use (transparent)
  - Has a routing hierarchy that lowers domain crossings
- Researchers may want to study more FPGA circuits…
  - Current SET research emphasizes compute units and inverter strings, but FPGA circuits (esp. routing) may behave differently
  - SET capture is a large component of error rates in designs that use hardened flip flops (e.g. DICE cell, SERT, 4TAG, etc.)
- Vendors may want to…
  - Make user friendly parts that serve more applications
  - Use RHBD with hardware TMR for a premium grade space part
  - Look into implementing no-domain crossing routing using place and route tools with existing parts