Dynamic Reconfiguration and Incremental Firmware Development in the Xilinx Virtex 5

J. Jones (jj4@princeton.edu)

Department of High Energy Physics
Princeton University

19th September 2008 / TWEPP 2008, Naxos

Outline

Introduction
- What's Happened to the FPGA?!

Structuring FPGA Firmware
- Standard Flow
- Pre-built IP

Pre-built Module Descriptions
- Post-Synthesis ngc/edif
- Post-Map/Place Macro
- Example: A Bus Macro

Dynamic Partial Reconfiguration
- Motivation
- Xilinx Virtex Reconfiguration
- Difference-Based Reconfiguration
- Difference-Based Example

There are several new ‘toys’ in the latest Xilinx FPGAs.

- Logic density has increased more than `ten-fold` compared to the generation of FPGAs commonly used in CMS.
- The typical maximum speed of the devices has also increased significantly.
- Clock routing, LUT structure and internal layout have changed completely.
- FPGA routing is an NP-hard problem, so a 10x increase in logic capacity results in a `significant` increase in build time.
- When using certain parts of an FPGA (e.g. the MGTs), the design may go from working to failure due to a routing change in the automatic tools (see G. Iles’ talk).

The number of hard IP blocks has also grown:

- Tri-mode Ethernet MACs.
- PCIe endpoints.
- System monitor with built-in ADCs (temperature / voltage).
- IODELAY controls, direct IOB clocking, integrated SERDES in IOBs.
- DCMs, PLLs, BUFRs, BUFGs.

This all adds up to a lot of confusion!

Typical Xilinx firmware flow in CMS at the moment is synthesise, translate, map, place and route.

- Design is (mostly) produced from VHDL or verilog + additional device-specific constraints.
- IP can be purchased as source, netlists or macros (see later) and introduced.
- Produces the best implementation, *but* due to the increasing size of FPGAs and firmware complexity, the turnaround time is growing rapidly (i.e. minutes $\rightarrow$ days).
- We now have such large devices that perhaps we can afford to be a little 'lazy'.

Custom IP can often be purchased / generated

- Part of design is already pre-built → reduce overall build time.
- We can do the same trick, if you know how...
- Post-synthesis → pre-built '.ngc/.edf' file (soft IP core).
- Post-map/place → pre-built '.nmc' file (partial hard IP core / macro).
- Post-route → pre-built '.nmc' file (full hard IP core / macro).

- Identify an individual firmware module in design with well-defined connections (e.g. Ethernet UDP stack).
- Synthesise design using e.g. XST.
- Ensure you turn of IOB pads (otherwise it thinks you are trying to instantiate the connections as pads on the FPGA).
- Product is an ngc/edf, which can be recombined with rest of design either pre- or post-final-synthesis (don't forget to match ngc name with component name).
- NOTE: As design is a netlist, final building of design may affect behaviour or module.

Post-Map/Place Macro

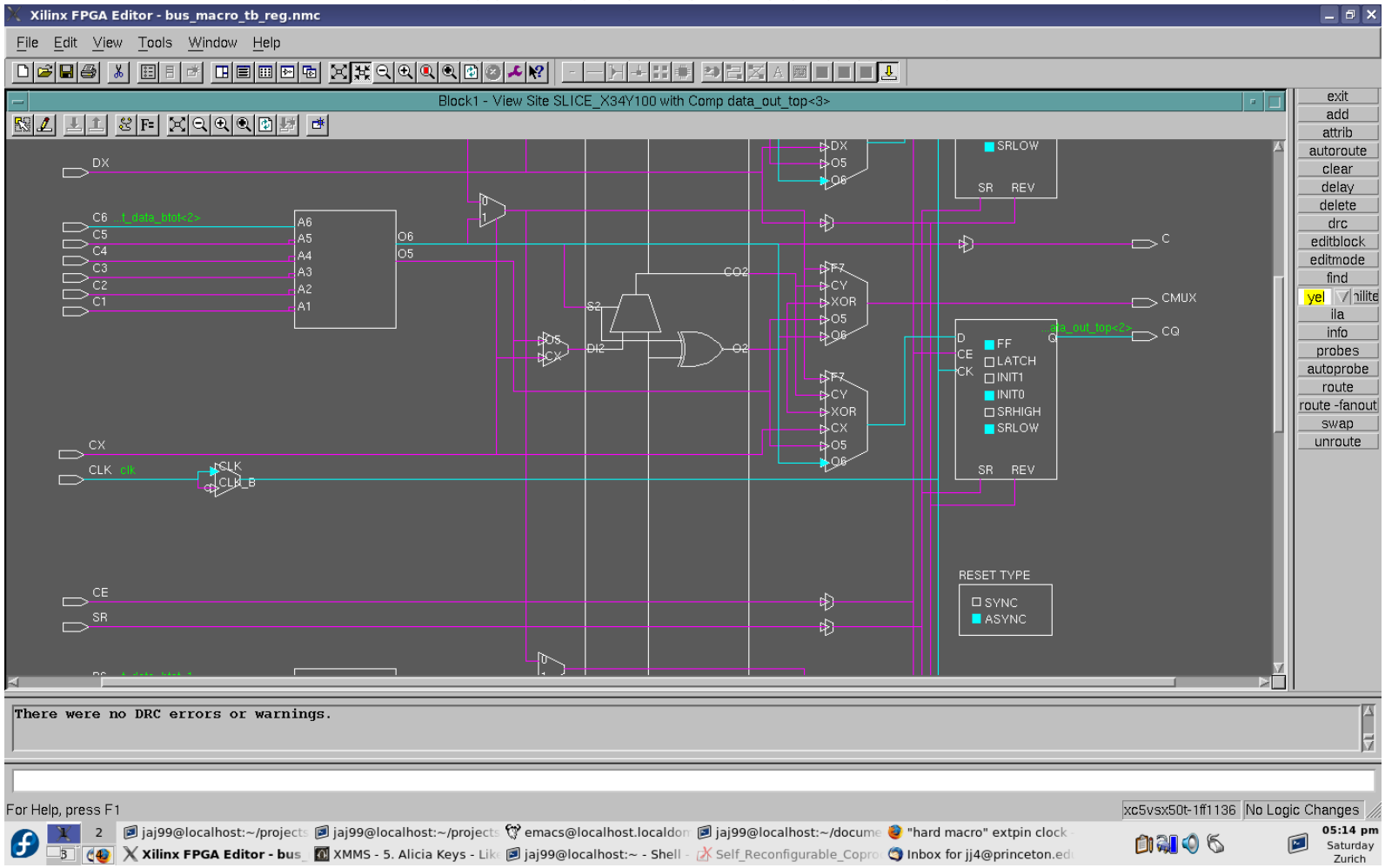
- Identify an individual firmware module in design with well-defined connections (e.g. Ethernet UDP stack).
- Synthesise, translate, map (and place) design with IOB pads and trimming turned off (don't forget to define timing!).
- Open design in Xilinx FPGA Editor, redefine design as a macro and add external ports + origin point.
- Origin defines a Relationally Placed Macro (i.e. RPM) - can be placed anywhere in chip with a matching layout.
- Product is an nmc, which can be recombined with rest of design during translate.
- CAVEAT: The use of routed macros can crash ISE (and no, Xilinx is not interested in fixing this).
- NOTE: They can be designed directly in Xilinx FPGA Editor or generated from a post-place and route netlist as well.

Bus macros are used as a static ‘bridge’ between partial designs in reconfigurable computing.

- Allows a static portion to of the design to modify a region of the FPGA at run-time by creating a fixed interface between regions.
- Can be used in DSP co-processing.
- Provides a useful example here as relatively simple.

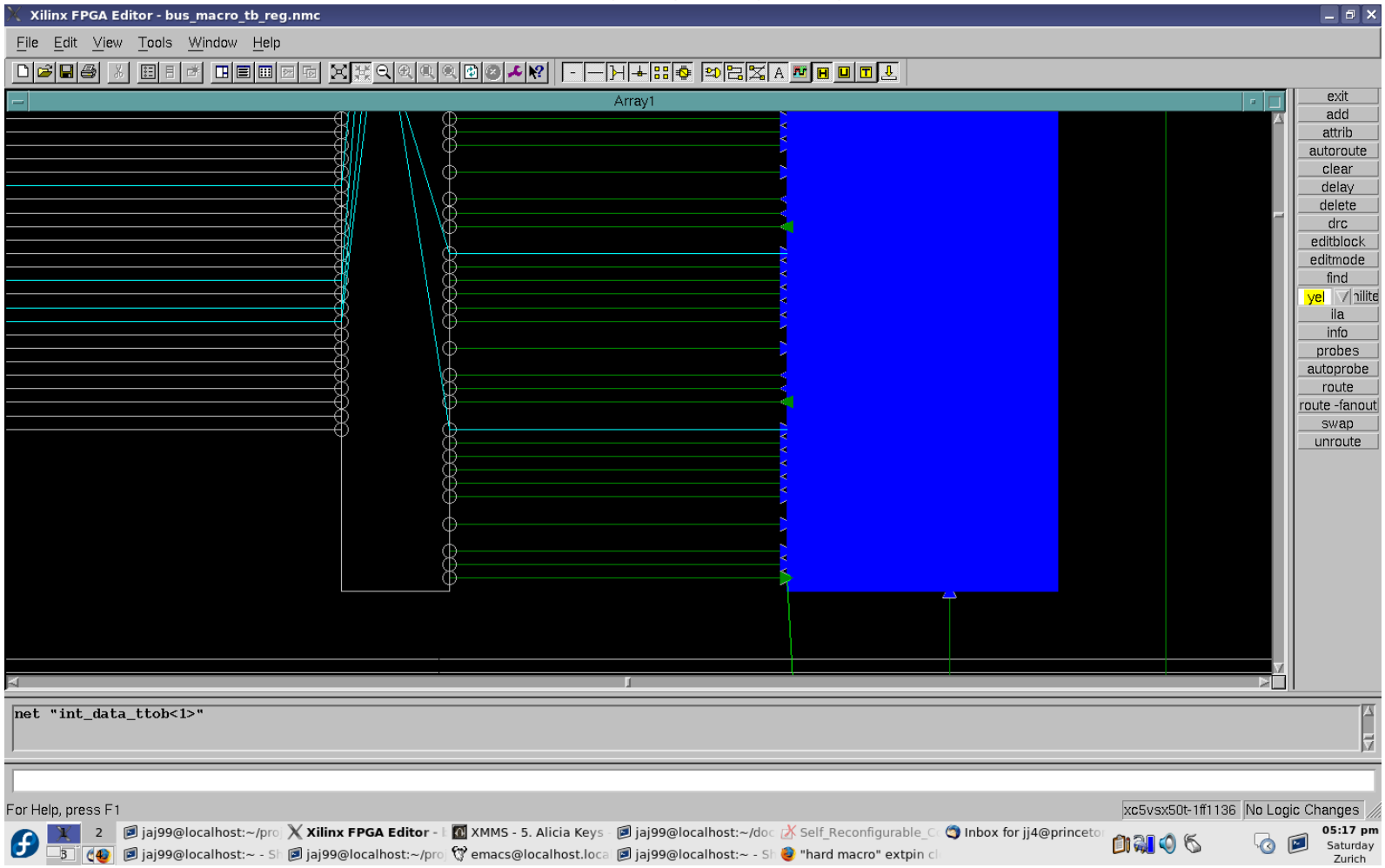
Example: A Bus Macro

Internal 6LUT structure



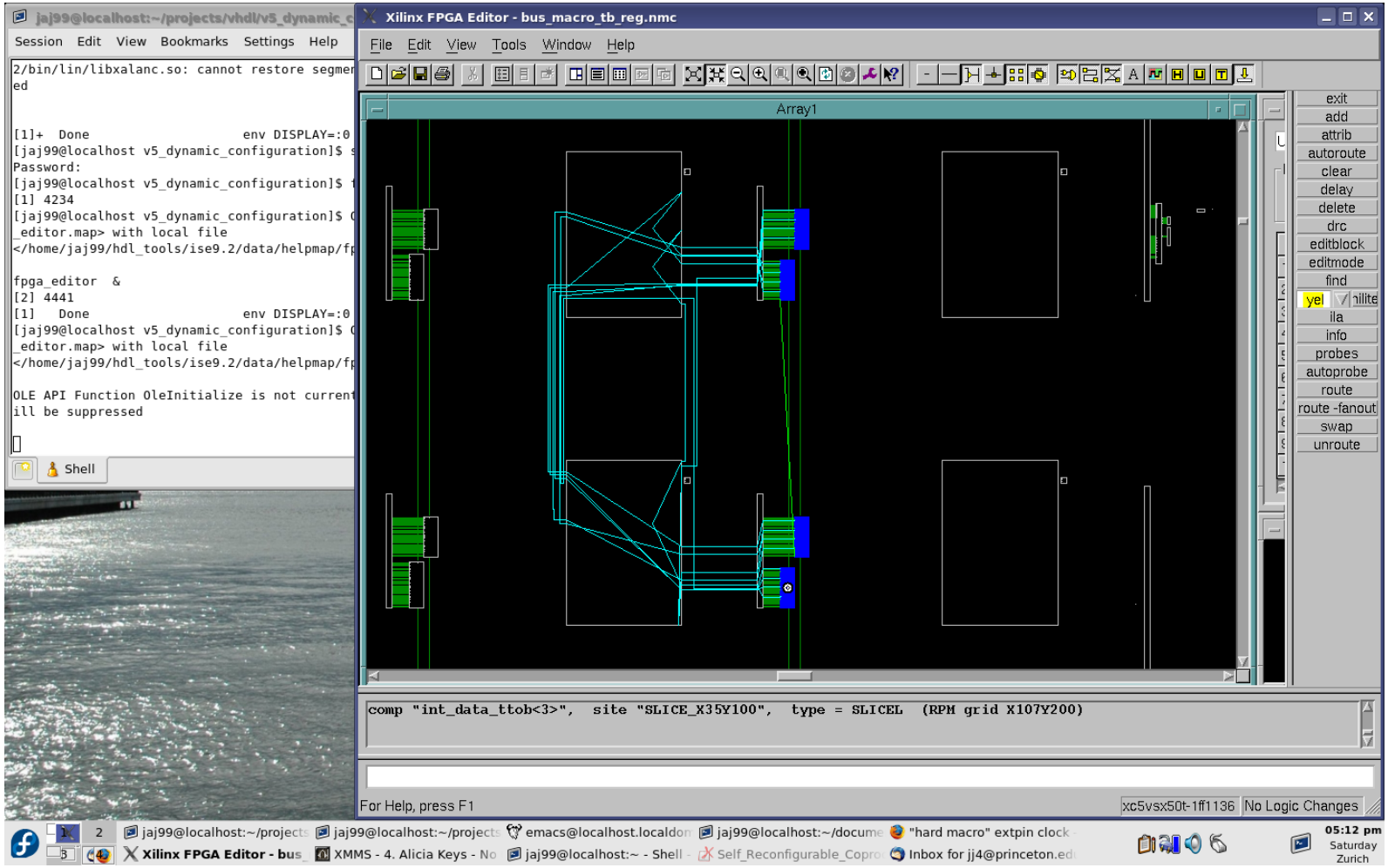
Example: A Bus Macro

A 4x6LUT SLICE



Example: A Bus Macro

A 16 LUT bus macro

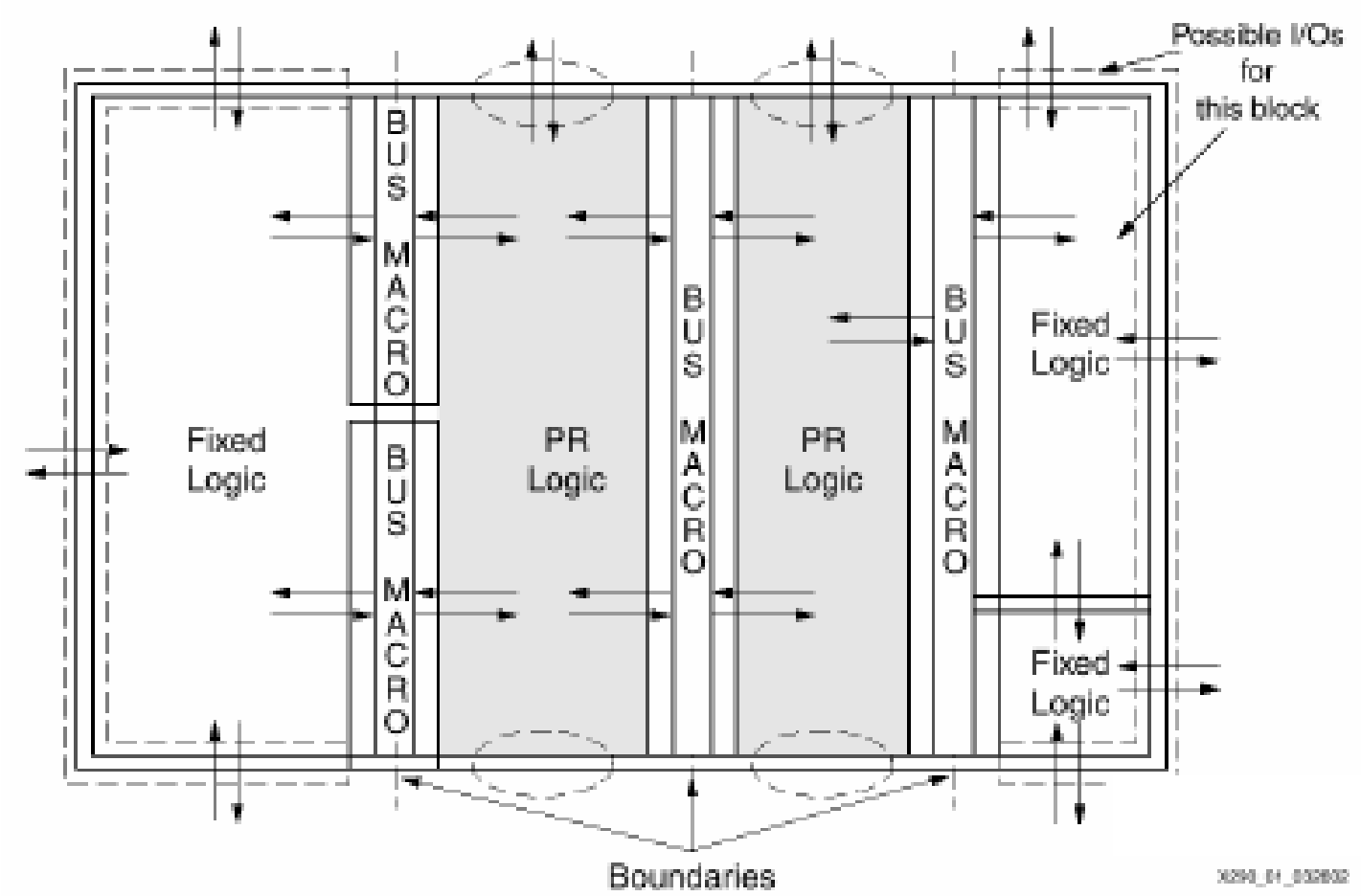


Many motivations for dynamic reconfiguration

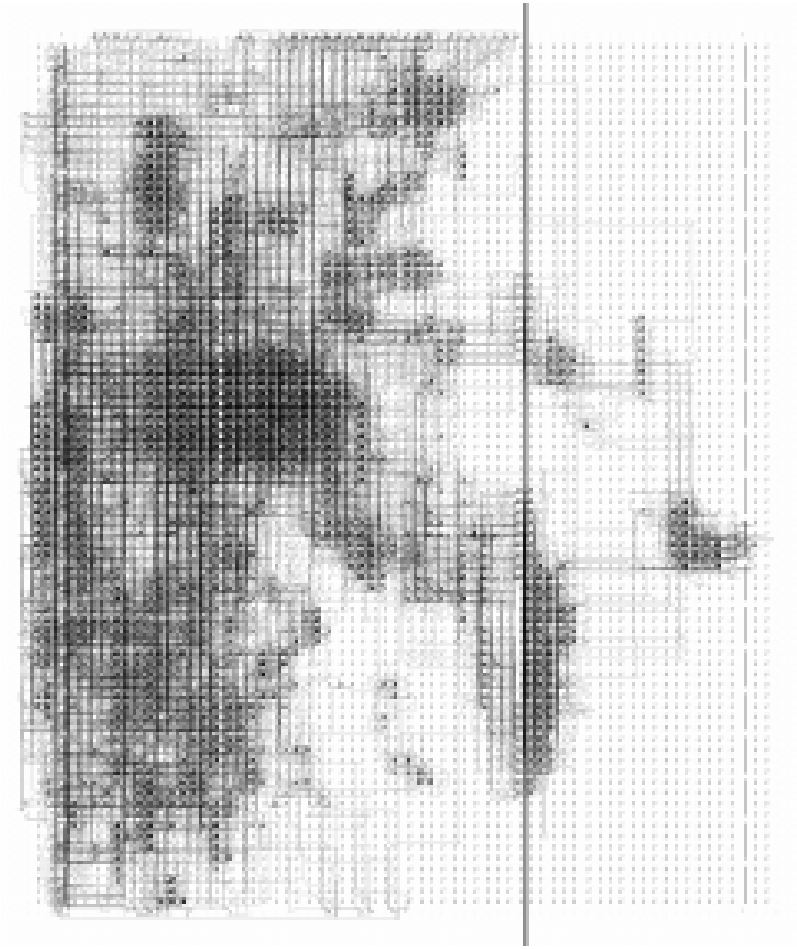
- Suppose you want to reduce device size and have two mutually-exclusive modes of operation (e.g. DES/AES encryption, software radio)...
- ...or you have no access to the device's configuration apart from the FPGA itself.
- Xilinx FPGAs contain an Internal Configuration Access Port (ICAP) which can access the SelectMAP interface.
- Can be used to do *anything the external interface can do.*

- Minimal reconfiguration unit of a Xilinx FPGA is a ‘frame’.
- These frames can be changed without powering down the device.
- If a portion of a frame does not change, the unchanged part is guaranteed not to glitch during reconfiguration.
- Master reset does not occur for initial logic states, so you *MUST* create your own reset circuit.

Reconfigurable areas (traditional approach)



Module-based design



- Instead of module-based (which is almost impossible in a Xilinx V5), what about difference-based?
- Just subtract one FPGA firmware from another ('bitgen -r').
- Problem - have to make sure the static part of the design is *identical* in both firmwares.
- Making fixed design literally identical is extremely difficult.
- A hard macro does not always work (you cannot use them for GTPs!).

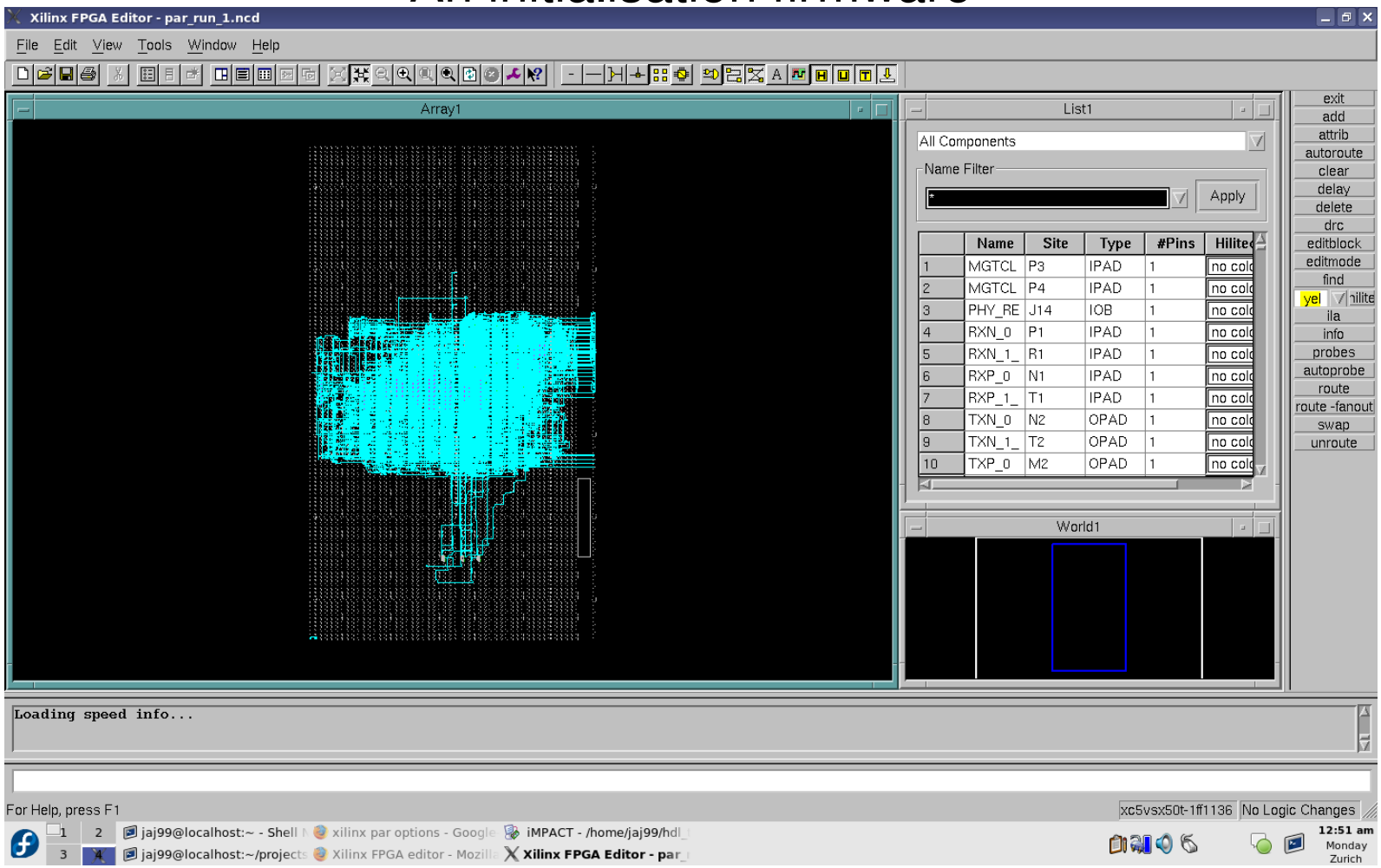
Method 1:

- Run synthesis without hierarchical cross-optimisation.
- Map, place and route design without logic trimming and exactly guide design using previous one as a reference.
- Fixed portion of design then becomes identical to original.
- Running ‘bitgen -r’ creates file with only differences between FPGA designs.
- NOTE: This method disappears in ISE 10 (exact guided P&R is not available).

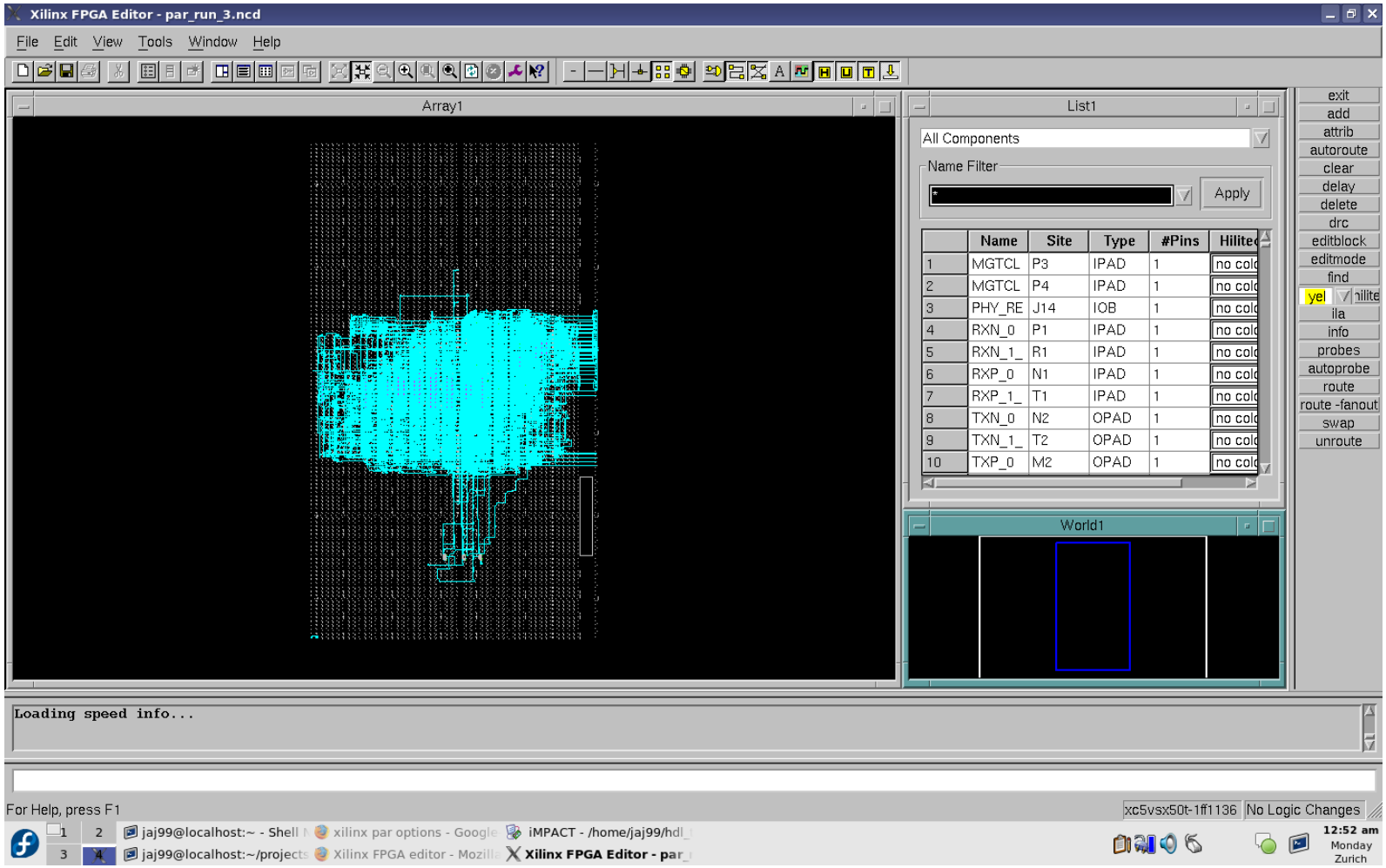
Method 2:

- Load fixed module in Xilinx FPGA editor and export the whole thing as a ucf constraints file.
- Run synthesis without hierarchical cross-optimisation.
- Map, place and route design without logic trimming and exactly guide design using the exported ucf.
- Fixed portion of design then becomes identical to original.
- Running ‘bitgen -r’ creates file with only differences between FPGA designs.

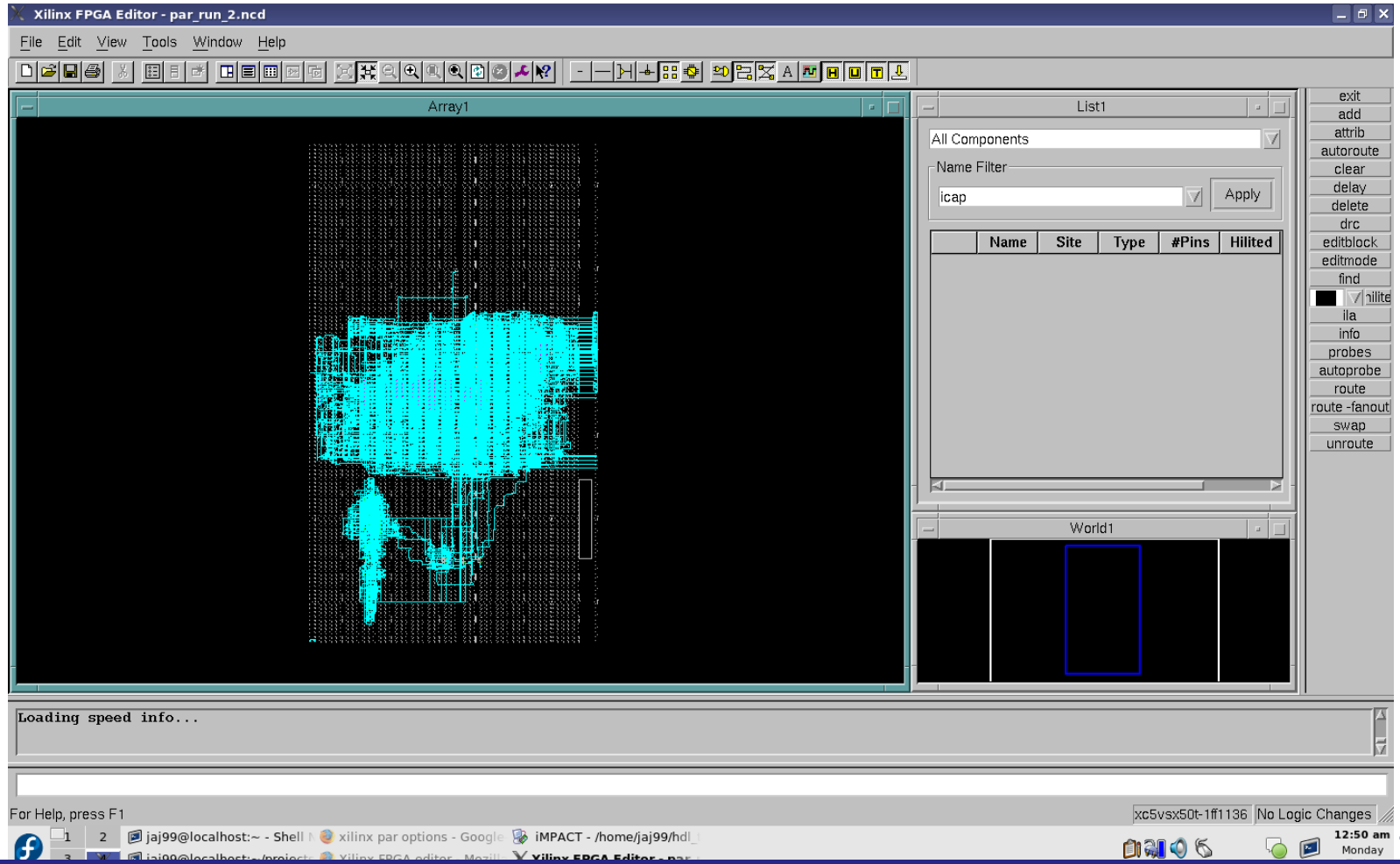
An initialisation firmware



Firmware modified to do loopback through bus macro

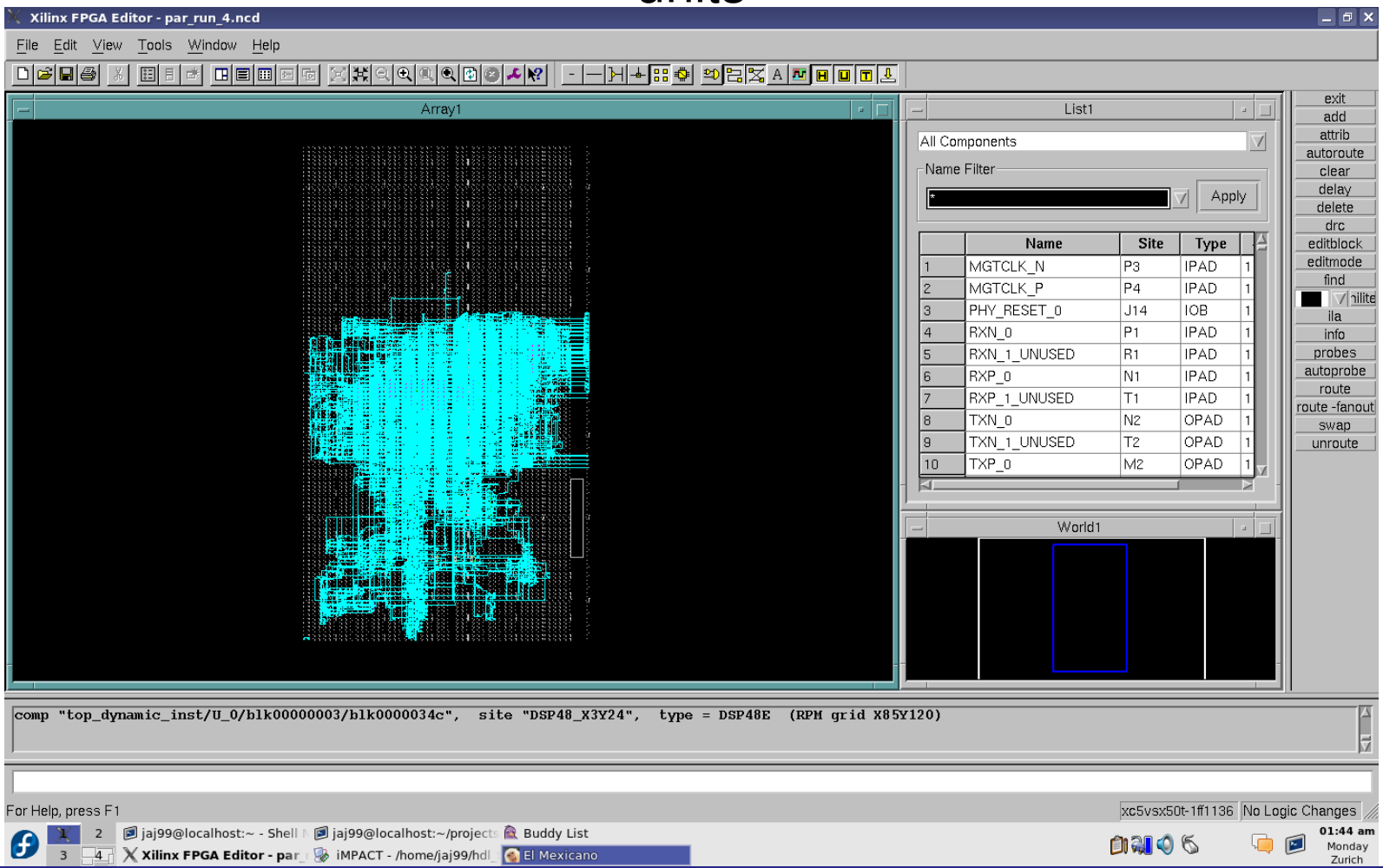


Firmware modified to add a double-precision floating point multiplier



DB Example

Firmware modified to add two double-precision floating point units



- Pre-synthesised netlists are easy to generate and require minimal turnaround time.
- Pre-placed/routed macros are also possible, but with caveats + relatively difficult to produce (i.e. you can't easily fix routing).
- Process is impossible to fully automate within ISE, but tools can be used on command line - use GNU Make.
- Can give you increased reliability with design re-use, and reduce turnaround time on large firmware projects.
- Dynamic reconfiguration is extremely tricky, but useful.
- The gap between the hardware capabilities and the capabilities of the software tools is increasing, which Xilinx needs to address.