SHAKTI

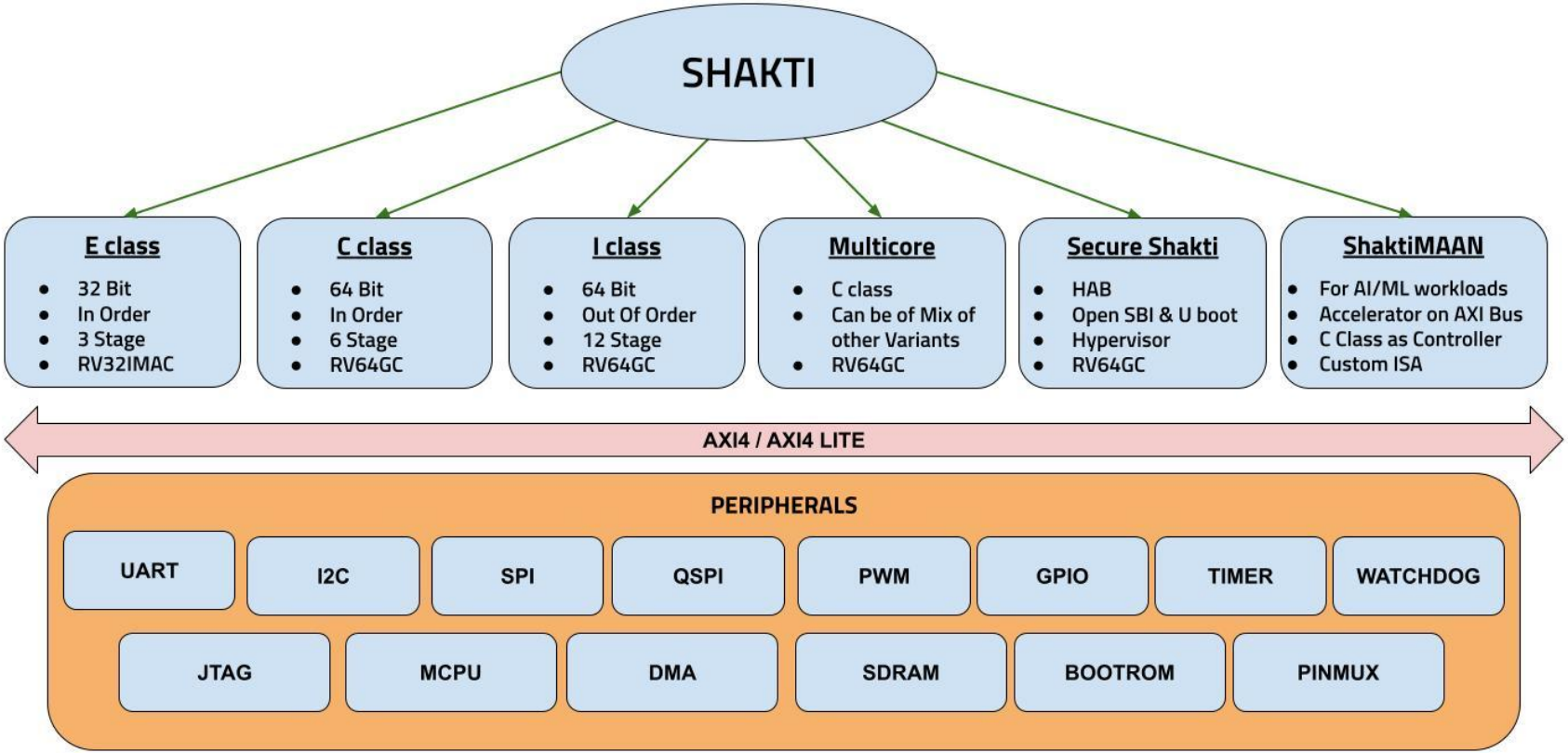
SHAKTI FPGA Design Flow

SHAKTI Group | CSE Dept | PS-CDISHA - RISE Lab | IIT Madras

Shakti Core

SHAKTI

SHAKTI Ecosystem



SHAKTI Design Portfolio

E Class vs C Class vs I Class

E-Class

- Operating frequency: Upto 400 MHz
- Positioned against ARM's Cortex Mx class cores
- Target: IoT devices, Edge Devices, Robotic Control, Smart cards

C-Class

- Operating frequency: > 1+ GHz
- Positioned against ARM's Cortex A35/A55
- Target: Reliable Computing, Secure Computing, IoT & Edge Computing hubs, Auto/Aerospace/Industrial Controls

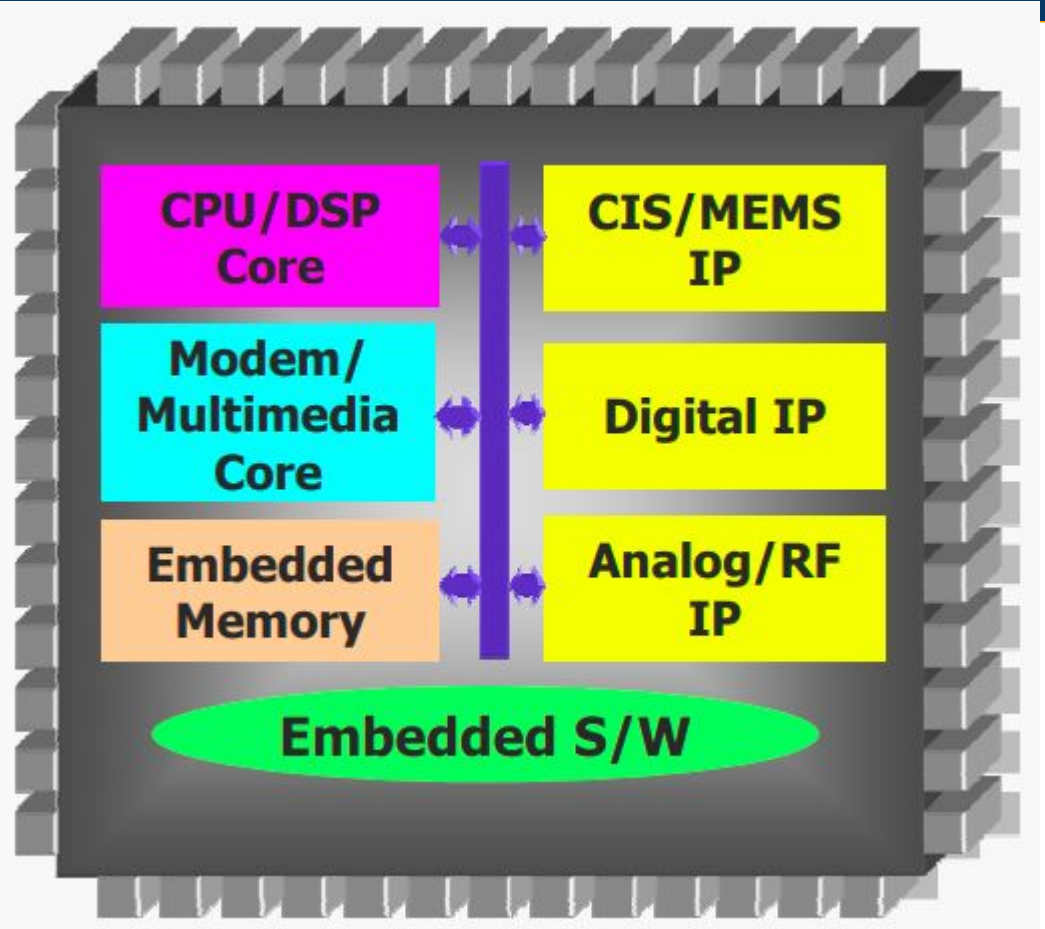
I-Class

- Operating frequency: 1-2 GHz
- Positioned against ARM's Cortex A76 (Work in Progress)
- Target: High Performance Compute, Mobile, Storage and Networking segments

SoC - System On Chip

SoC

- Integration of a complete system by replacing a product/application which needs multiple chips to a single IC.
- Very Large transistor counts on a single IC.
- Mixed Technology on a same chip (digital protocols, memory, analog, etc.).



Why SoC

- Complex applications.
- High performance.
- Battery Life.
- Short market window.
- Cost sensitivity.
- Modern technologies.

SoC Applications

- Telecommunications (Ethernet switch, bridge, router, ATM switch, etc.)
- Portable Consumer Products (MP3 players, Display systems, Mobiles, etc.)
- Multimedia (camera, games, Video).
- Embedded Control (Automotive, printers, smart cards, etc).

FPGA – Field Programmable Gate Arrays

What is an FPGA?

- **Field Programmable Gate Array (FPGA)**
- **Reconfigurable Hardware** – FPGAs enable dynamic hardware customization.
- **HDL Programmability** – Designed using HDLs to model chip behavior.
- **Post-Deployment Adaptability** – Modify functionality even after deployment.

FPGA Technology Giants

AMD (Xilinx)[1]

Intel (Altera)[2]

Microchip (Microsemi)[3]

[1] https://www.amd.com/en/products/adaptive-socs-and-fpgas/fpga.html

[2] https://www.intel.com/content/www/us/en/products/details/fpga/arria.html

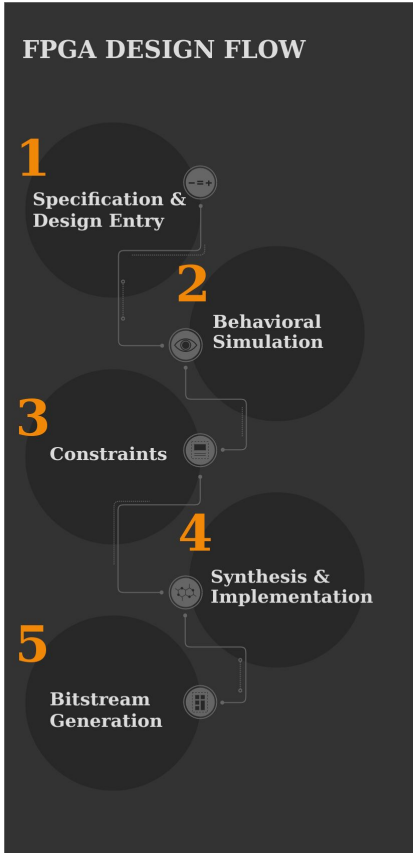
[3] https://www.microchip.com/en-us/products/fpgas-and-plds/fpgas

SHAKTI

Why FPGA

- ASIC - Application specific Integrated Circuit i.e. IC designed for a specific application.
- ASIC fabrication is a tapeout
- A costly process which involves millions of Dollars i.e. huge cost investment.
- Verification process is a must.
- Simulation & FPGA verification.
- Simulation - Will not guarantee on timing.
- FPGA - will provide both timing & functional validation i.e. at low frequencies.

FPGA Design Flow



FPGA Design Flow

- **Specification & Design Entry:** Getting the requirements/specifications & low/high level design.
- **Behavioral Simulation:** Writing the code (verilog, vhdl, system verilog & BSV).
- **Synthesis:** Converting the hdl to a netlist i.e. list of logic elements with connections along with optimization.
- **Simulation:** verification of the specification with simple test benches.

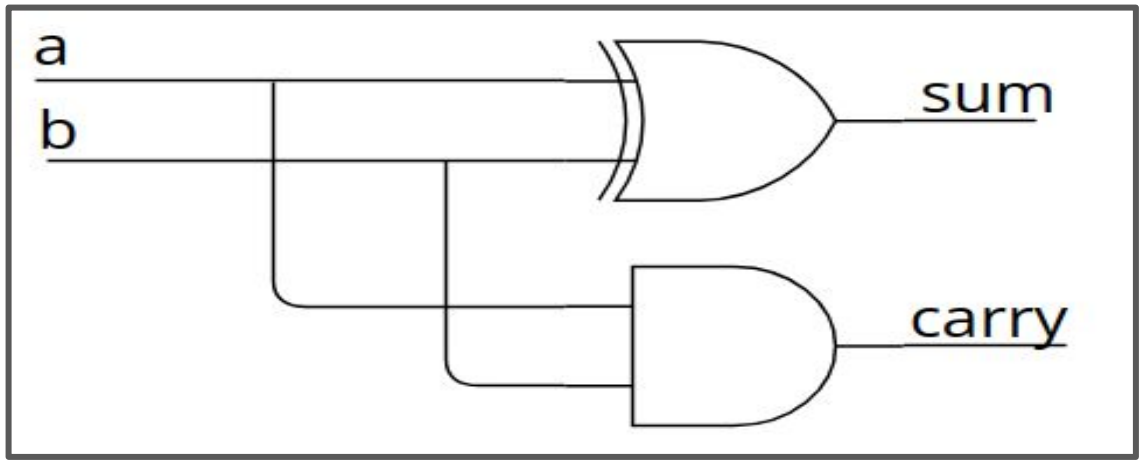
FPGA Design Flow

- **Implementation:** Process of converting the netlists to FPGA specific pattern
  - **Translate** - collects and merge netlists into a single net list & verify the constraints.
  - **Map:** Mapping the resources with net lists
  - **Place & Route:** physical placement of the netlist to the FPGA physical resources
- **Bitstream generation:** converting the implementation design into a FPGA understandable format.

A Step-by-Step Walkthrough in Nexys Video

1. Design Entry - HDL Coding (Verilog or VHDL or System Verilog)

```
module half_add (
     input a,b,
     output sum,carry
);
assign sum = a ^ b;
assign carry = a & b;
endmodule
```



A Step-by-Step Walkthrough in Nexys Video

2. Behavioral Simulation

A Step-by-Step Walkthrough in Nexys Video

3. Project Device (Board Selection)

A Step-by-Step Walkthrough in Nexys Video

4. Constraints
(Xilinx Design Constraints)

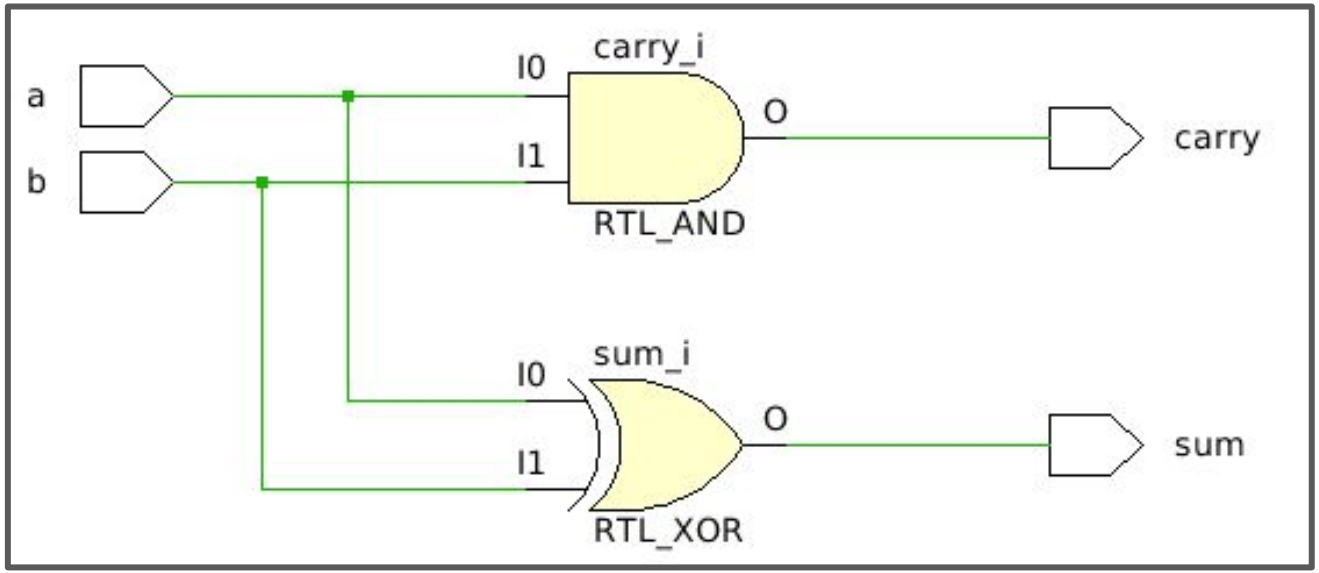
```
## Switches
set_property -dict { PACKAGE_PIN E22  IOSTANDARD LVCMOS12 } [get_ports { a }]; #sw[0]
set_property -dict { PACKAGE_PIN F21  IOSTANDARD LVCMOS12 } [get_ports { b }]; #sw[1]

## LEDs
set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS25 } [get_ports { sum }]; #led[0]
set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS25 } [get_ports { carry }]; #led[1]
```

Inputs are mapped to **switches** & **Outputs** are mapped to **LEDs**.

SHAKTI

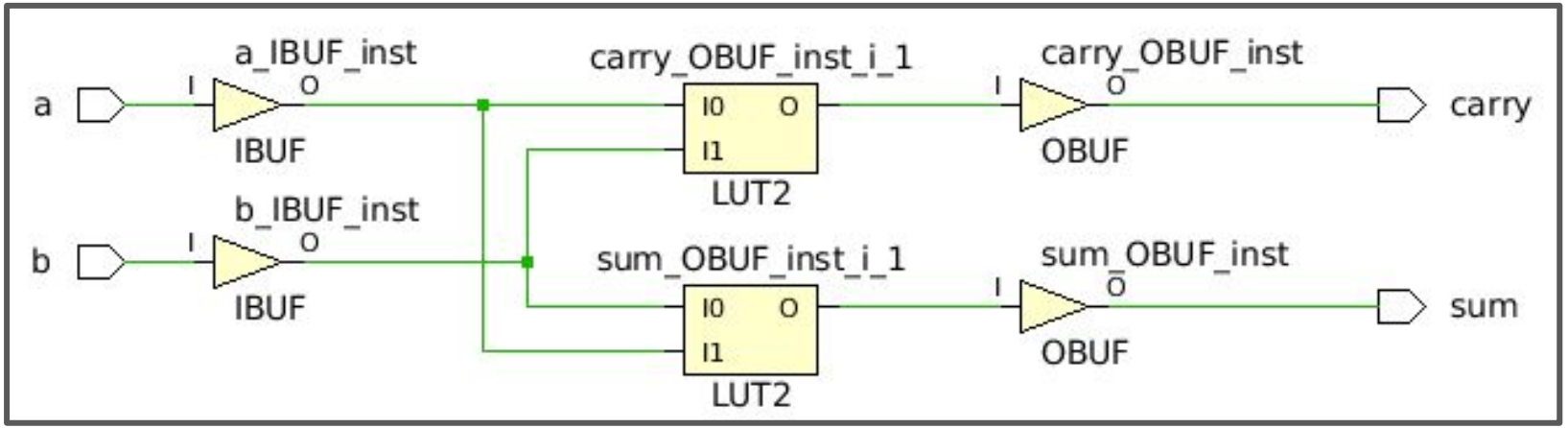
A Step-by-Step Walkthrough in Nexys Video

5. Synthesis (Translation)



A Step-by-Step Walkthrough in Nexys Video

6. Place & Route (Implementation)



Each **LUT** will have **6 inputs** in **Nexys Video**.

A Step-by-Step Walkthrough in Nexys Video

LUT Utilization

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	1	133800	0.01
IO	4	285	1.40

Vivado steps

- Create Project
- Select Device/Board
- Speed,Language
- Add source Codes and constraints.
- Add Vivado IPs.
- Synthesis the design.
- Implement the design.
- Generate bit stream
- Program hardware.

Shakti Core FPGA flow features

- Project creation & build are tcl script and Makefile based.
- Make is customizable for different boards.
- ISA & peripherals are configurable in yaml files.
- TCL scripts aids in instantiate the required IPs, creating the project, build the project,program the mcs file.
- Codes are written in bluespec.
- Simulation support.
- Peripheral instances and address mapping are configurable in bluespec files.
- IP porting has few custom procedures.

SHAKTI

Shakti FPGA flow Steps

- Clone the repo.
- Configure ISA, Core & debugger in yaml files.
- Select the board.
- Make instances of peripherals and the address mapping.
- Get the dependency repos.
- Compile the bluespec code to create the verilog files.
- Instantiate the required Vivado IPs.
- Create the project.
- Build the project.
- Generate mcs file.
- Program the mcs.
- Use the FPGA as Shakti.

SHAKTI

Shakti Steps

Steps are simplified:

- Clone the code (https://gitlab.com/shakti-iitm/shakti-projects/sos.git).
- Run the make command “make build BOARD=arty_a7_ganga”.

Shakti IP porting

SHAKTI

Shakti IP Porting

- Adding peripherals to Shakti core.
- IP can be
  - Available Shakti based Slow peripherals IPs.
  - Available Vivado IPs.
  - Adding new verilog code with Shakti core.
- Few predefined steps needs to be followed for smooth IP integration.

References:

Shakti Documentation: https://shakti.org.in/documentation.html

Shakti Blogs: https://blogshakti.org.in/

Shakti FPGA files: https://gitlab.com/shaktiproject/sp2020

Shakti SDK: https://gitlab.com/shaktiproject/software/shakti-sdk

Shakti on Arduino:
https://blogshakti.org.in/how-to-print-hello-world-on-shakti-using-arduino-ide/

Linux on Shakti: https://gitlab.com/shaktiproject/software/linux-on-shakti

Platform IO: https://registry.platformio.org/platforms/platformio/shakti

Shakti cores: https://gitlab.com/shaktiproject/cores

Shakti peripherals: https://gitlab.com/shaktiproject/uncore/devices

Queries

SHAKTI

Thank you!

Website: shakti.org.in

GitLab: gitlab.com/shaktiproject