



### IP-XACT extensions as common object file for HLS and System Integrator

An electronic datasheet for design portability: each FU (leaf IP block) is described in extended IP-XACT.

Includes a wide-coverage, parameterisable net-level protocol model:

- A port has one direction of data, but a port in the other direction may piggy-back on its handshaking arrangements with implicit timing.
- Encompasses pipelined, fully-pipelined, put-aout, AXI, locallink.

Enables black boxes to be wrapped up and deployed (CAM example).

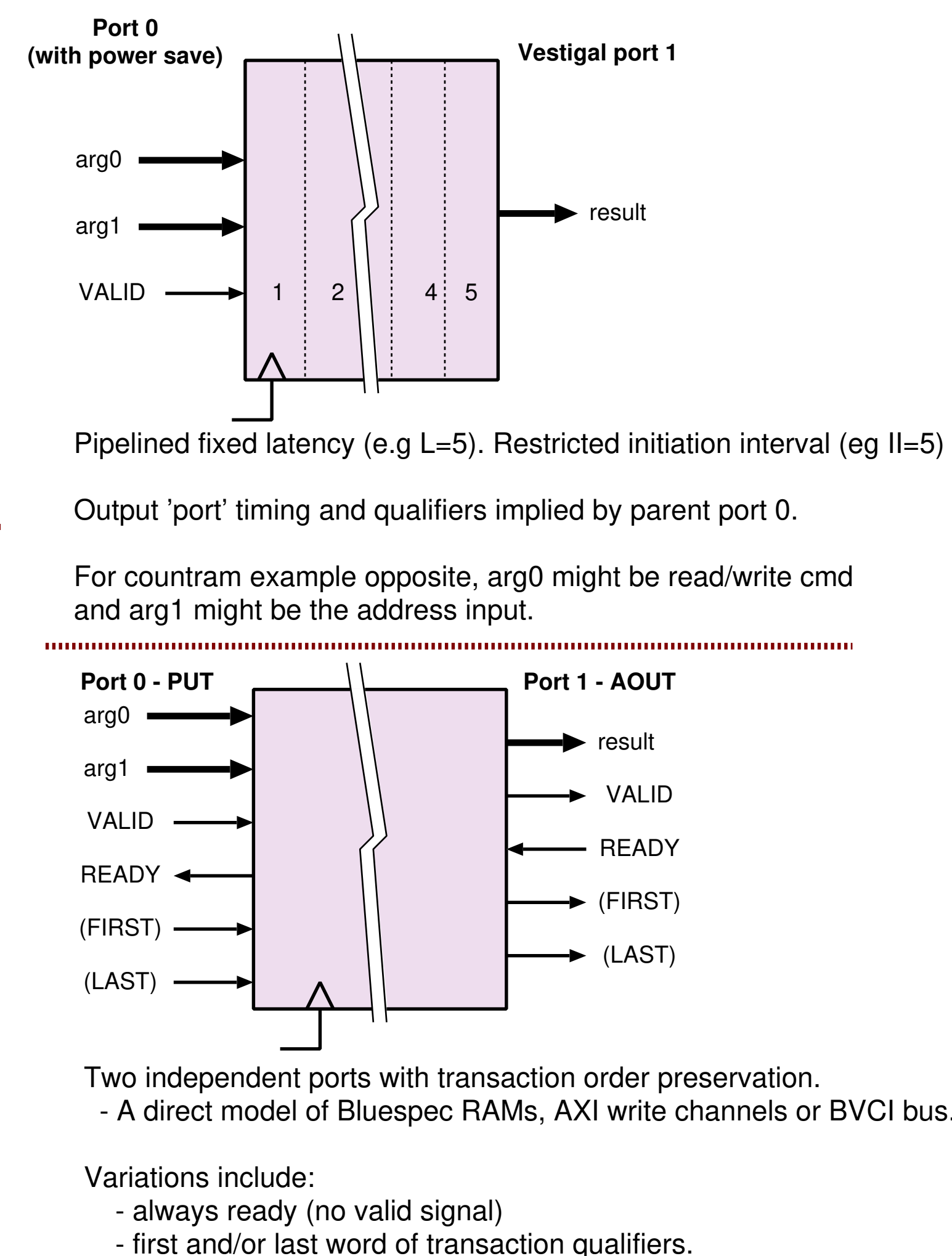
Two main uses:

- Incremental HLS - use a prior compilation inside a current one,
- Multi-FPGA deployment auto partition and inter-wiring generation.

What is documented:

- Net-level port-structure and formal model of per-port protocol,
- TLM method set for each port (for HLS),
- Whether freely portable between FPGA or locally bonded out,
- Whether stateless (so can be freely mirrored for load balancing),
- Combinational delays for pipeline design in parent,
- Sequential dependencies documented (for deadlock avoidance),
- Claim space needed in off-chip DRAM memory,
- Mutually-exclusive/non-reentrant entry points in TLM model,
- EIS (an end in itself) tainting to avoid logic trimming,
- Expected area, power and other non-functional attributes,
- GUI icon and URLs for further data,
- (License cost/conditions!)

Specification drafts can be downloaded from QR code top right.



### Incremental HLS Toy Example: Network Traffic Monitor

Download all files from above QR code.

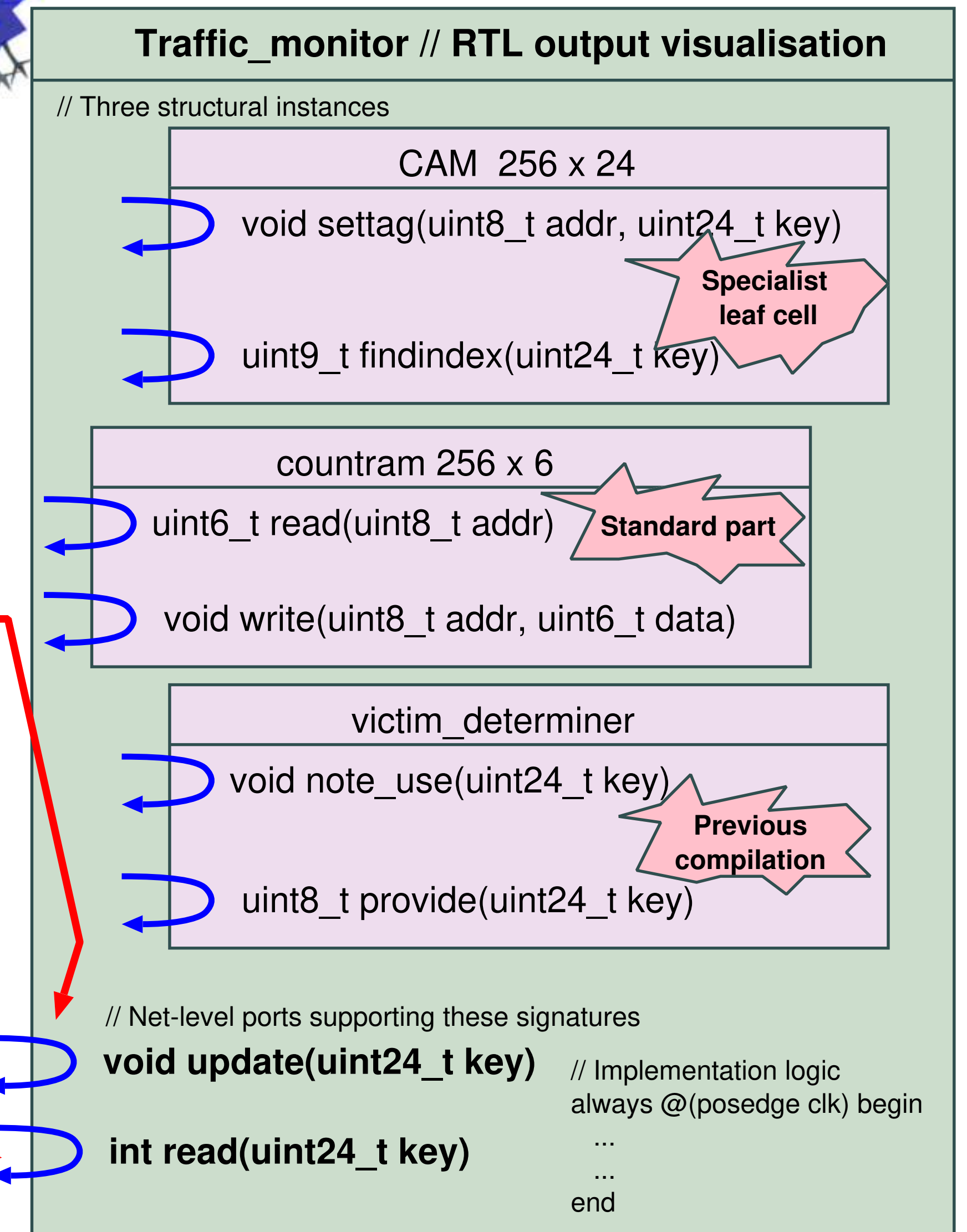


```
class traffic_monitor // Source code for B/P HLS
{
  CAM256 *cam = new CAM256(); // Content-addressable
  uint6_t countram[256]; // Random access
  VD *victim_determiner = new VD(); // LRU/random?

  public void update(uint24_t key)
  { int idx = cam.findindex(key);
    if (idx < 0)
    { idx = victim_determiner.provide(key);
      cam.settag(idx, key);
      countram[idx] = 1;
    }
    else countram[idx] = MIN(63, countram[idx]);
    victim_determiner.note_use(key);
  }

  public int read(uint24_t key)
  { int idx = cam.findindex(key);
    return (idx<0) ? -1: countram[idx];
  }
}
```

This OO/transactional definition of a simple traffic monitor uses three pre-existing/pre-compiled FUs and exports two methods (actually the nets for invoking them). Generated RTL can preserve the block instance sub-structure and later flattened for inter-block optimisations.



The instantiated FUs might use a variety of interface standards and latencies. Compilers (eg B/P) adapt, supporting a variety of standards for the imported FUs and exported methods.

### Multi-Blade FPGA Allocation: Tool Flow

All IP-blocks and pre-compiled FUs are described with extened IP-XACT.

Rule-directed System Integrator Planner tool takes:

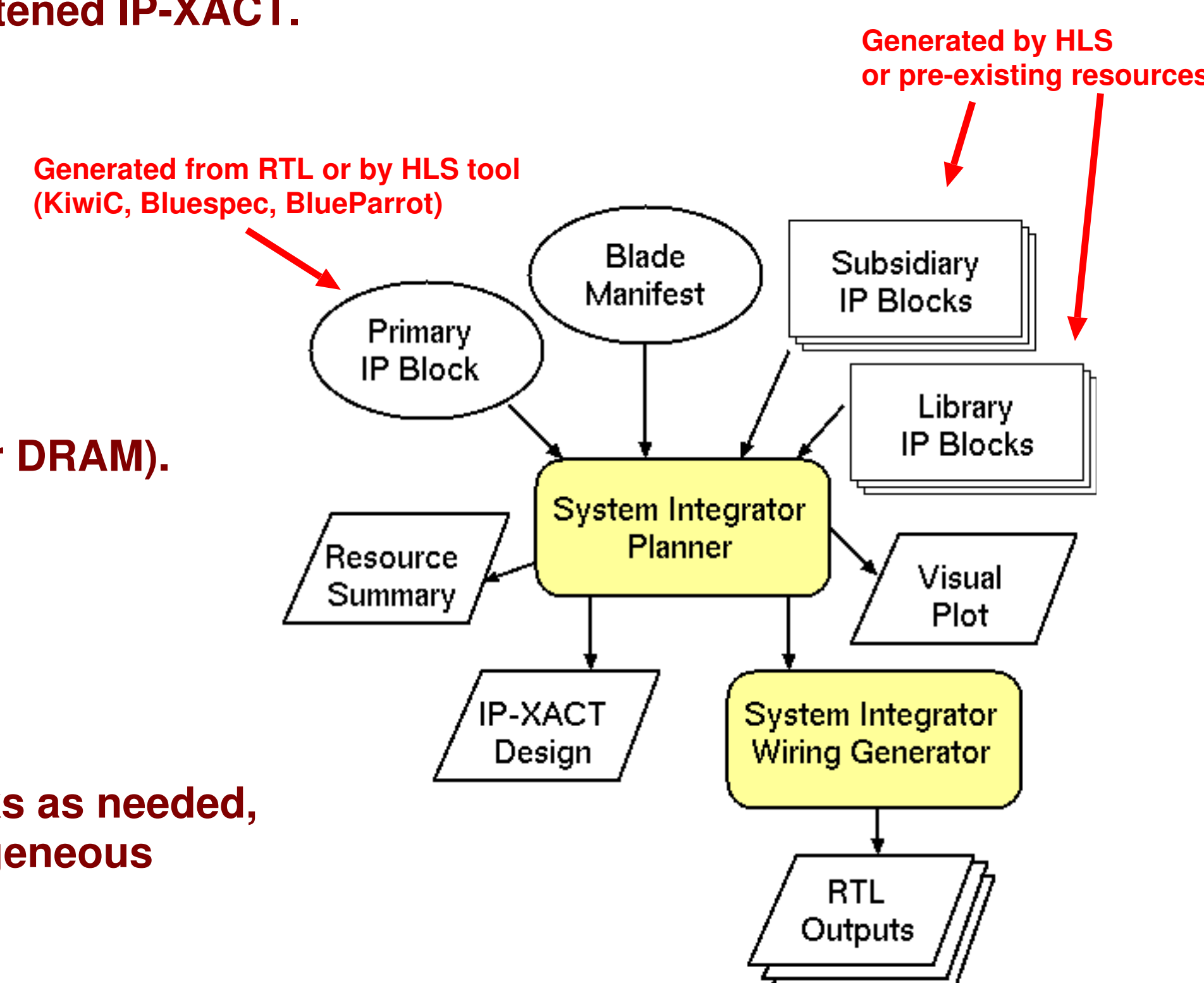
- Top-level application IP block (or blocks)
- An indexed library of pre-compiled sub-systems
- A 'blade manifest' giving:
  - The number and sized of FPGAs available,
  - Interconnection topology,
  - Number of DRAM banks and sizes for each FPGA,
  - Hardwired resources available - (eg video output or DRAM).

Planner functions:

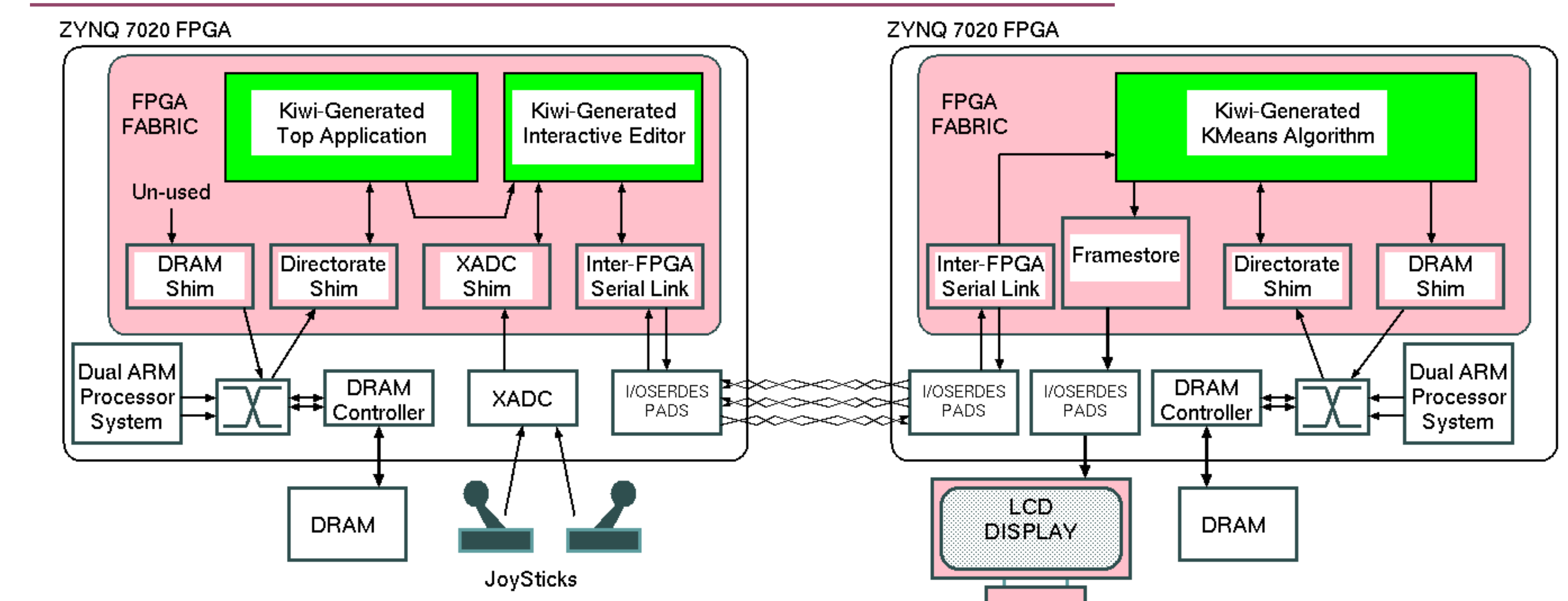
- Infers/optimises the required multiplexors and other bus infrastructure components,
- Implements debug network as needed,
- Provides each component with shared and private heap space in common memory or off-chip DRAM banks as needed,
- Automatic deployment of protocol adaptors for heterogeneous port interconnection and to use inter-FPGA links.

Wiring generator emits:

- a top-level RTL file for each FPGA (or ASIC)
- a further IP-XACT file reporting the aggregate (for incremental implementation of this step too!).



### System Blade Partition and Assembly Example (FPL-17 Kiwi)



FPL-17 demo flow:

Five or six C# programs were manually fed to KiwiC (via Make)  
Three project-specific RTL files resulted - each with IP-XACT metadata.  
System Integrator combined these with existing IP blocks and wrote two toplevel files, one for each FPGA.

Everything touched is recompiled, and TCL/batch Vivado invoked, giving new bitfiles (all still under Make).

