SpaceWire Working group 11th Nov 2004

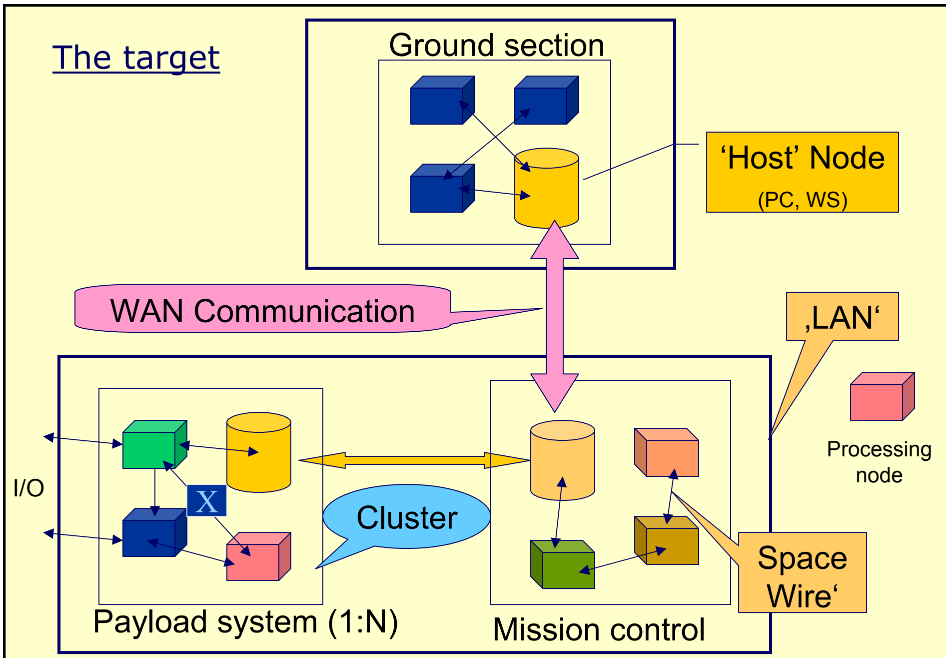
SpaceComRTOS

A distributed formal RTOS adapted to SpaceWire enabled systems

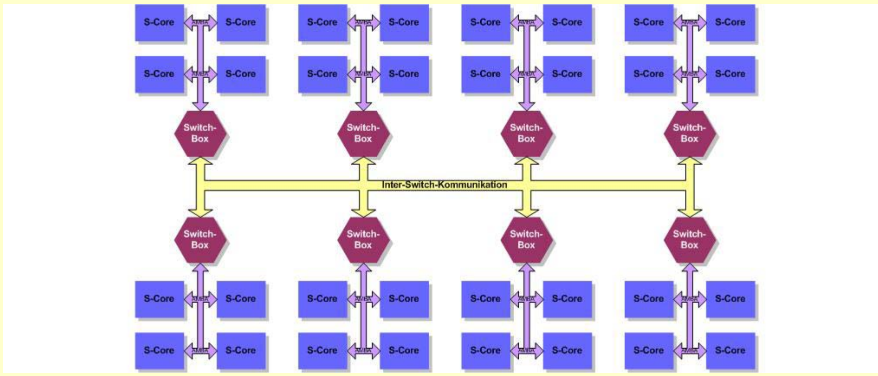
Eric.Verhulst@OpenLicenseSociety.org

www.OpenLicenseSociety.org

The target



MP-SoC target



- No general solution on the market !
- Severe memory constraints
- Needs : low latency, high bandwidth, use few resources
- FPGAs with softcores can put lowest layers in logic fabric

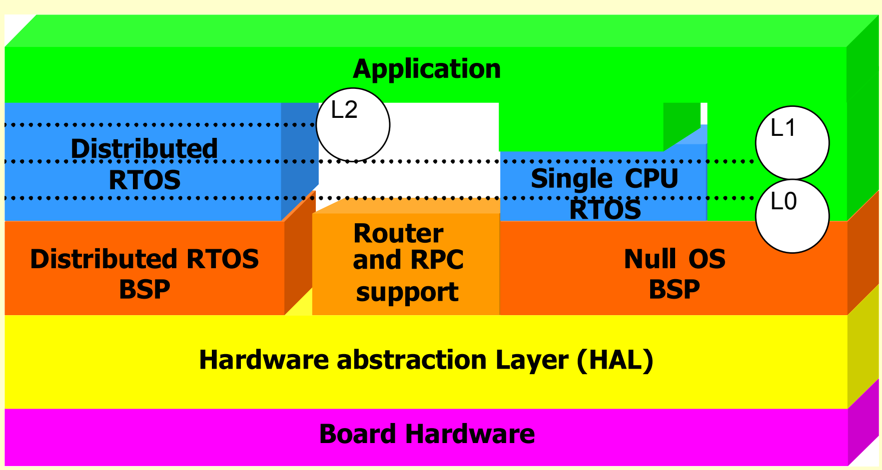
Core functionalities (applying to distributed embedded systems)

- Multi-tasking (or: processes, threads, ...)
  - Best paradigm for executing multiple functions on same processor
  - Provides modularity and information hiding for software development
  - Needed to reduce idle time while waiting for communication
  - Sleep modes during idle time saves power
  - Allows concurrency with system level support
- Scheduling
  - Static only for synchronuous dataflow
  - Dynamic for meeting real-time constraints
- Synchronisation
  - With hardware
  - Between tasks
- Communication
  - Idem
  - Application independent link drivers
- Memory management
- Resource management

Some requirements

- Portability:
  - across processor types
  - across communication backbones
  - across heterogenuous systems
  - cross-development on host
- Scalabilty:
  - from SP to MP to loosely coupled
  - local support as well WLAN support
  - user extensible
  - topology independent:
    - build routing and buffering
    - ‚distributed semantics'
- Performance
  - real-time scheduling and real-time communication
  - low latency, low memory foot print
- Reliability
  - correctness by design
  - ‚trust-worthy component'

General architecture



OpenComRTOS

- Scalable distributed RTOS based on message passing
  - actually: **scalable communication** layer with **scheduling** support
- distributed semantics (transparant parallel programming)
- formally analysed and validated
  - extensive system-wide message passing protocols
  - using formal model checkers (e.g. based on CSP)
- safety and security by compatible plug-ins/extensions
  - same external behavior, but blocking ‚faults' and ‚intrusions'
- 3+ layers :
  - (NULL-OS)-L: testing and local I/O
  - LO
    - very small (1 K), core primities, core system packects
    - typical use : MP-SoC, DSPs
    - includes scheduler, low latency router and drivers
  - L1
    - sema, queue, mailbox, resources, ...: traditional RTOS services
    - emulate RTOS (but often only SP), cabinet level
  - L2
    - supports widely distributed operation, RT-CORBA

OpenComRTOS

OpenComRTOS: Level L2

- variable size packets
- widely distributed adressing
- dynamic protocol packets
- extensible API

• public connections
• target : 100-500K

OpenComRTOS: Level L1

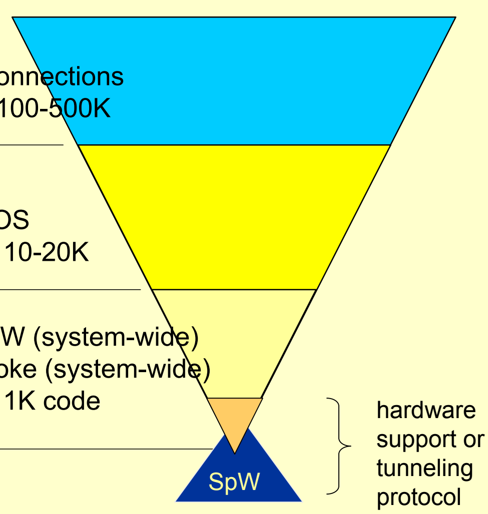
- fixed size packets
- cluster adressing
- dynamic protocol packets
- API emulation

• std RTOS
• target : 10-20K

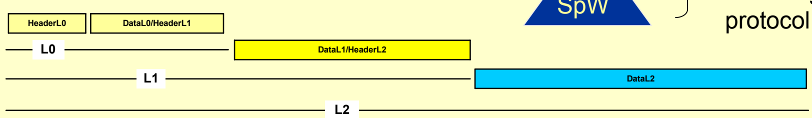
OpenComRTOS: Level L0

- fixed size packets
- tightle clustered adressing
- static protocol packets
- system packets
- scheduler
- runtime monitor

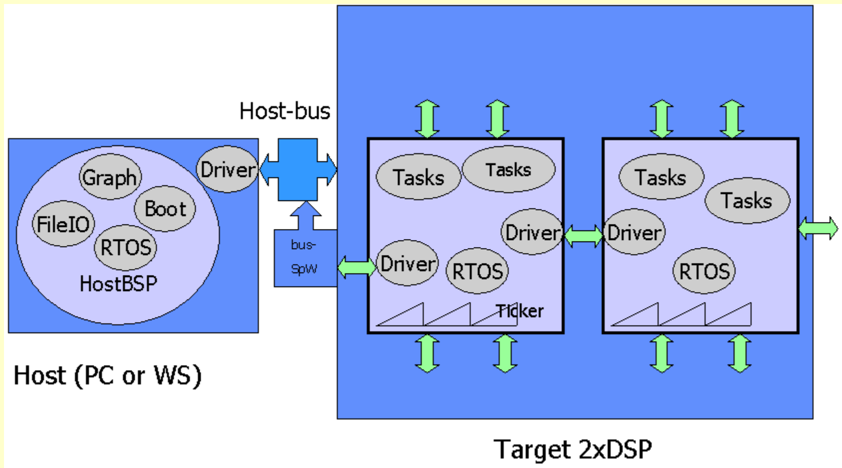
• semaMW (system-wide)
• peek-poke (system-wide)
• target : 1K code



Packet structure



Development and target environment



-

11-11-2004 9

Different protocols: permanent presence

- Boot packets
- Command packets
- Data packets
- Debug packets
- Logging packets ?
- I/O packets ?
- ... NOT TOO MANY !
- Each protocol has ‚sub-states‘ (sequence-chart)
- Protocols must be hierachical, reflected in datastructures
- All can map on protocol identifiers
- Standardisation needed for interoperability

11-11-2004 10

NULL-OS layer

- Hardware semi-dependent layer: no scheduling
- Important for diagnostics, debugging, booting phase
  - minimum interference with OS en hardware
  - not intended for operational use
- Worm: map out existing topology
- Netloader: boot complete system via links
  - from ‚host' (mission controller) or safe mass-memory
  - processors
  - peripherals (INT's, SMCS, ...)
  - flash, eprom, ..
  - option: floodfiller
- I/O with hostserver during development or during operation
  - access to hostservices and ports
  - can be remote over e.g. ground connection
- Not limited list of services: see document
  - Board and processor specific variations
  - Standardisation helps in portability

L0

- Minimum ‚RTOS'
  - scheduling
  - routing
  - buffering
- Minimum set of primitives
  - SemaSetMW
  - SemaGetMW
  - Peek (remote or local read)
  - Poke (remote of local write)
  - or just a Move (distributed memcopy) ?
  - SetScheduling
- Target code size: 1K

L1

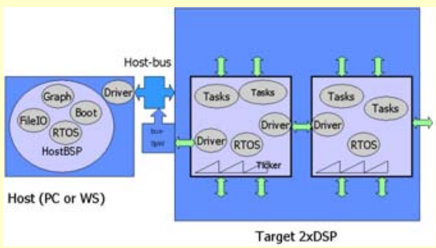
- Higher level, traditional RTOS services
  - but most RTOS semantics are not suitable!
- Local events (binary)
- Distributed:
  - counting semaphores
  - queues
  - mailboxes
  - pipes
  - memory maps
  - resources
  - process control
- Group operations
- Many-to-many semantics
- Blocking, non-blocking, time-outs
- Emulation of COTS RTOS (within limits)
- Target codesize: 10-20K

L2

- Mostly open
- Primitives to define new packets and protocols
  - CreatePacket
  - SendPacket
  - ReceivePacket
  - DefineService
  - CallService
- Should allow to run across heterogenuous, ‚alien' networks using tunneling

(Link) Drivers

- Three types:
  - Point-to-point: between directly connected nodes, no protocol
  - Direct-to-I/O: between node and I/O, I/O specific protocol
  - NetLink drivers: to provide virtual connections with system-level protocol
- Error recovery and faults
  - make maximum use of SpaceWire support (rather unique in world!)
  - principle:
    - layered transaction protocols with ACKs
    - failures and errors at lowel level are invisible at higher levels:
      - build-in safety support, but as an option
    - allows to program application independently of fault support mode



System wide addressing issues

- In order to provide transparency at runtime, each destination of source should have unique identifier
- Tree-structered domains, scope issues
- Cfr. IPv4 or IPv6 addressing
- What can have an address or logical ID ?
  - task or processor
  - I/O port
  - link port
  - host node
  - ...

Remaining issues

- Trade-offs remain a developer‘s issue. Low level programming still possible but to be used with care
- Support for static operation vs. dynamic operation
- Trade-off between portability, performance, development time and development reliability
- Likely not possible or even desired that all hardware features are supported (possible conflicts)
- Better a simple design that works than a complex one (with many features) that is not fully predictable
- Better a simple design that is a bit slower on average than one that is faster some of the times