



The Gannet Service Manager: a Distributed Dataflow Controller for Heterogeneous Multi-core SoCs

Wim Vanderbauwhede, Paul McKechnie,
Chidambaram Thirunavukkarasu

Department of Computing Science
University of Glasgow, UK

Institute for System-Level Integration
Livingston, UK



Overview



- Heterogeneous Multi-Core SoCs
- Gannet SoC architecture
- Design implementation of the Service Manager
- Performance



Heterogeneous Multi-Core SoCs

- SoCs with multiple heterogeneous IP cores
- NoC for decoupling of computation from communication
- For reconfigurable systems, the **control** aspect is crucial
- Gannet SoC architecture: framework for run-time task reconfiguration
- This work: design and implementation of the Gannet Service Manager



The Gannet service-based architecture

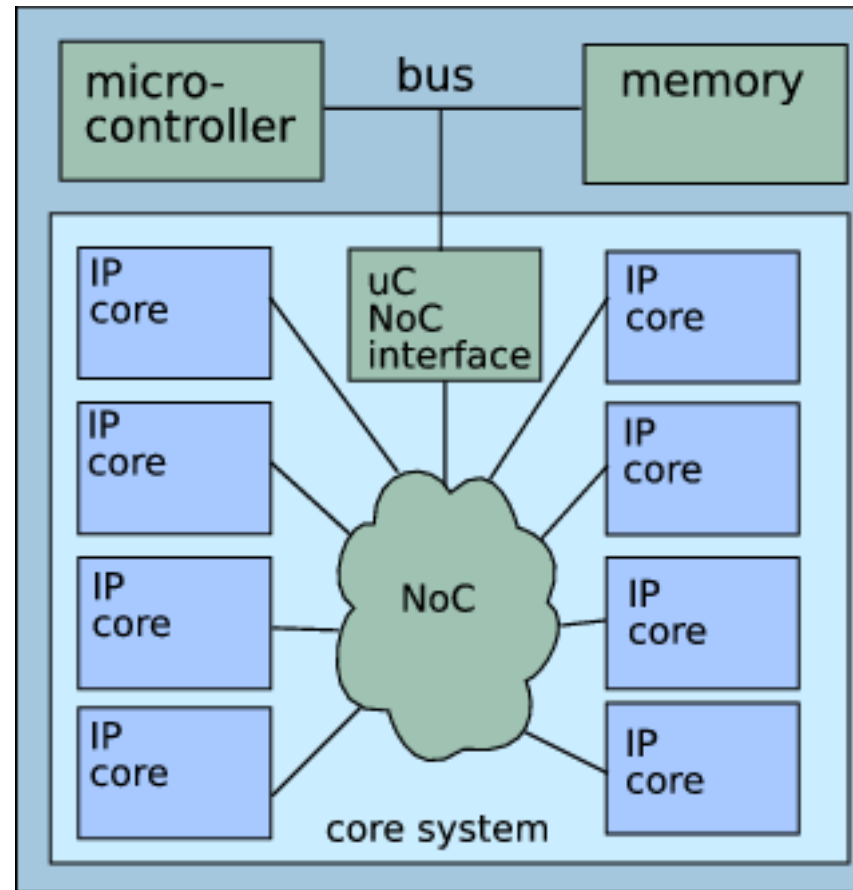


This **service-based SoC architecture** framework consists of three core components:

- a task graph description language with compilation toolchain
- a packet-based switched communication medium (NoC)
- a novel control circuit interfacing with each IP core (the Service-Manager)

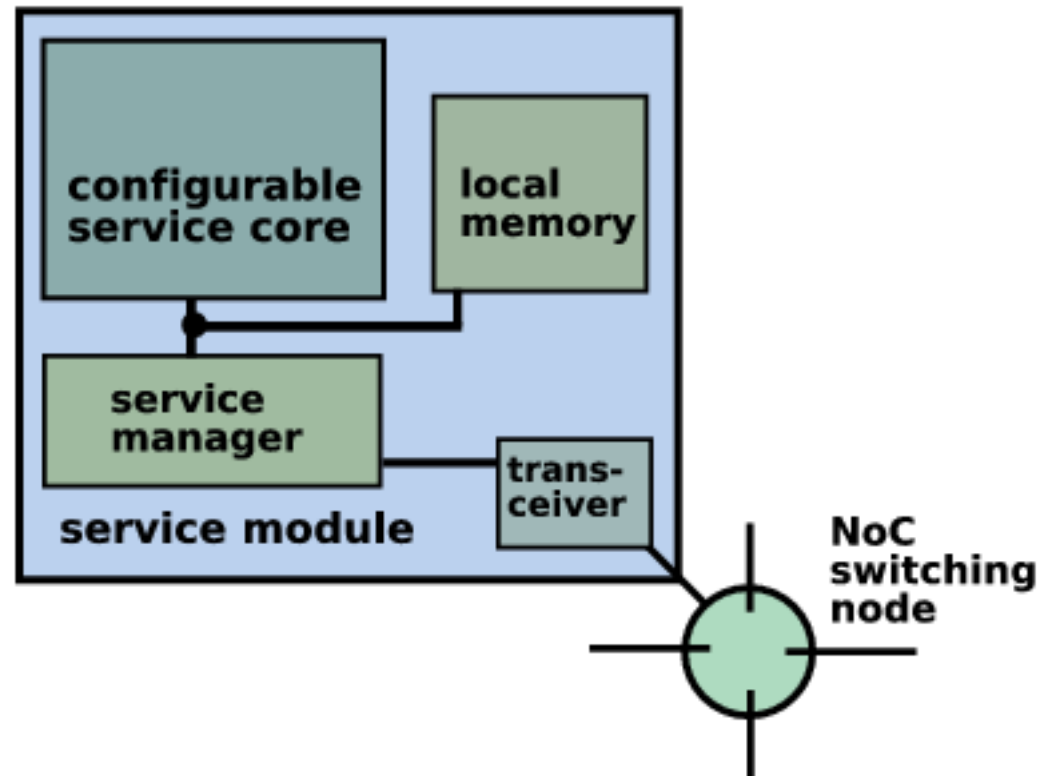


NoC-based SoC





SBA Tile





Functional programming model

Gannet design philosophy: exploit the full potential for concurrency offered by massively multi-core systems

- Computations should be executed as much as possible in parallel rather than sequentially
- Every IP core in the system provides one or more services.
- A service can be thought of as a function: it consumes data and produces data.
- Functional task graph composition: all data transfers are the results of calls to services.



The Gannet Service Manager



The Gannet Service Manager

- Third-party IP cores don't act as a services (not aware of the rest of the system).
- Generic interface for controlling the dataflow in a heterogeneous multi-core System-on-Chip (SoC).
- Small, high-performance specialised processing unit aimed at NoC-based SoCs.
- Coarse-grained distributed computing platform for heterogeneous IP cores
- Full control over the data path.



The Gannet language

- Assembly language for the Gannet machine
- Intermediate language syntax: S-expressions (cf Scheme)
- All arguments of a function are evaluated in parallel
- Call to a service S : $(S\ a_1\ a_2\ \dots\ a_n)$.
- Trivial example a SoC with 5 services:
 - `image`: image capture from several cameras,
 - `compose`: creating a composite image
 - `convert`: conversion to jpeg format or png format
 - `compress`: compression
 - `encrypt`: encryption



The Gannet language

- a compressed composite of raw image from cameras 1 and 3:

```
(compress  
  (compose  
    (image 'camera1)  
    (image 'camera3)  
  )  
)
```

- a jpeg-converted, encrypted image from camera 2:

```
(encrypt  
  (convert 'jpeg  
    (image 'camera2)  
  )  
)
```



Adding Control



■ a conditional branch

```
(let
  (assign 'img1 (image 'camera1))
  (if (< (size (read 'img1)) 'max_sz)
      '(convert 'png (read 'img1))
      '(convert 'jpeg (read 'img1)))
  )
)
```



Compilation

- Gannet system is fully packet-based
- Programs are compiled into code packets
- Compilation process replaces nested expressions by references.
- The example

```
(encrypt  
  (convert 'jpeg  
    (image 'camera2)  
  )  
)
```

is compiled into 3 packets:

```
r0⇒(encrypt r1)  
r1⇒(convert 'jpeg r2)  
r2⇒(image 'camera2)
```



Gannet system operation

Gannet system: a set of services connected via a NoC

The simplified operation sequence is:

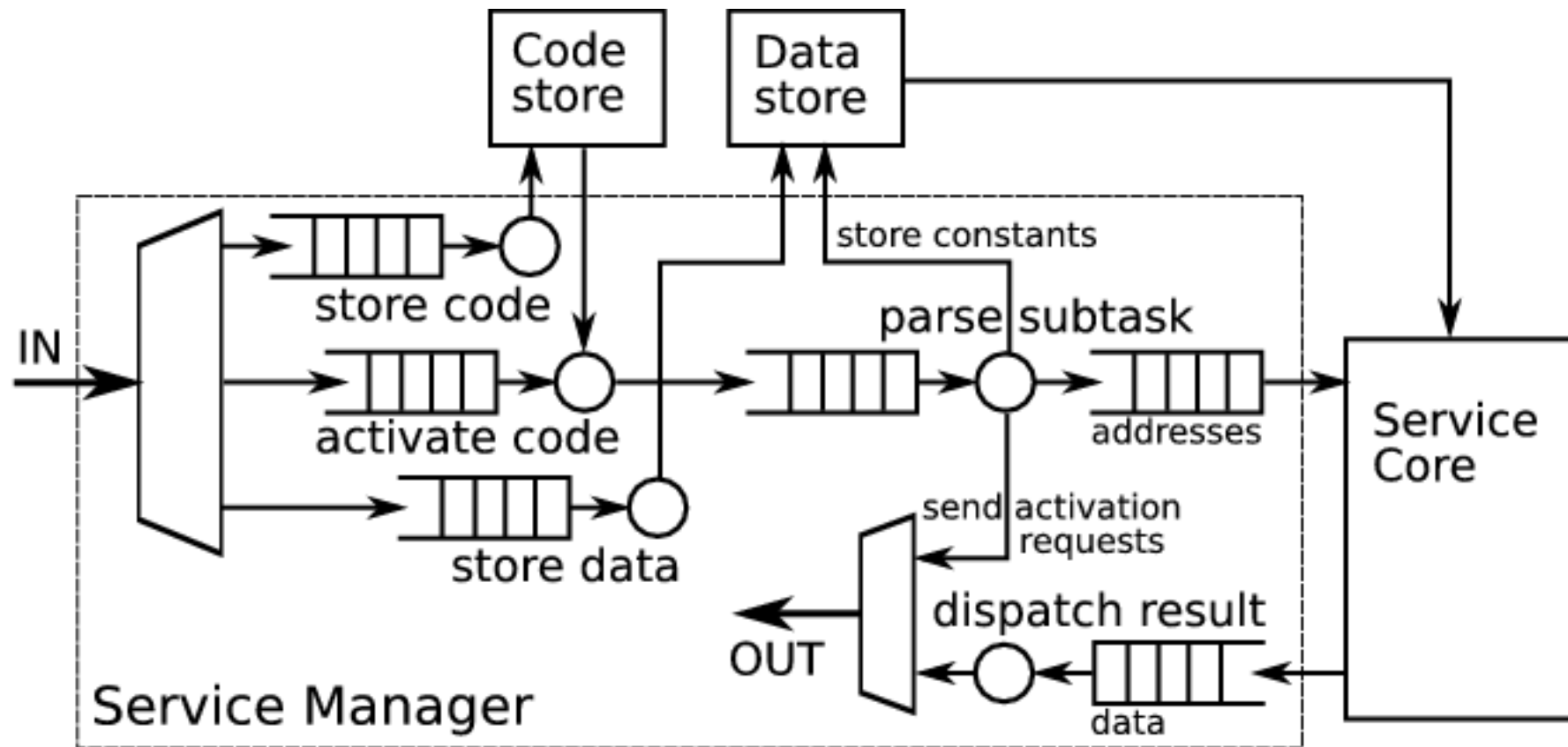
- initially, all service managers receive one or more **code** packets
- when a service managers receives a **reference** packet, it activates the subtask.
- execution of this subtask results in dispatch of reference packets to other subtasks
- leaf tasks (tasks without any references) are executed by the service core and the result packets are returned.

This operation sequence results in a fully parallel execution of all branches in the program tree in an unspecified order governed by the processing time of the packets.



Service Manager Implementation

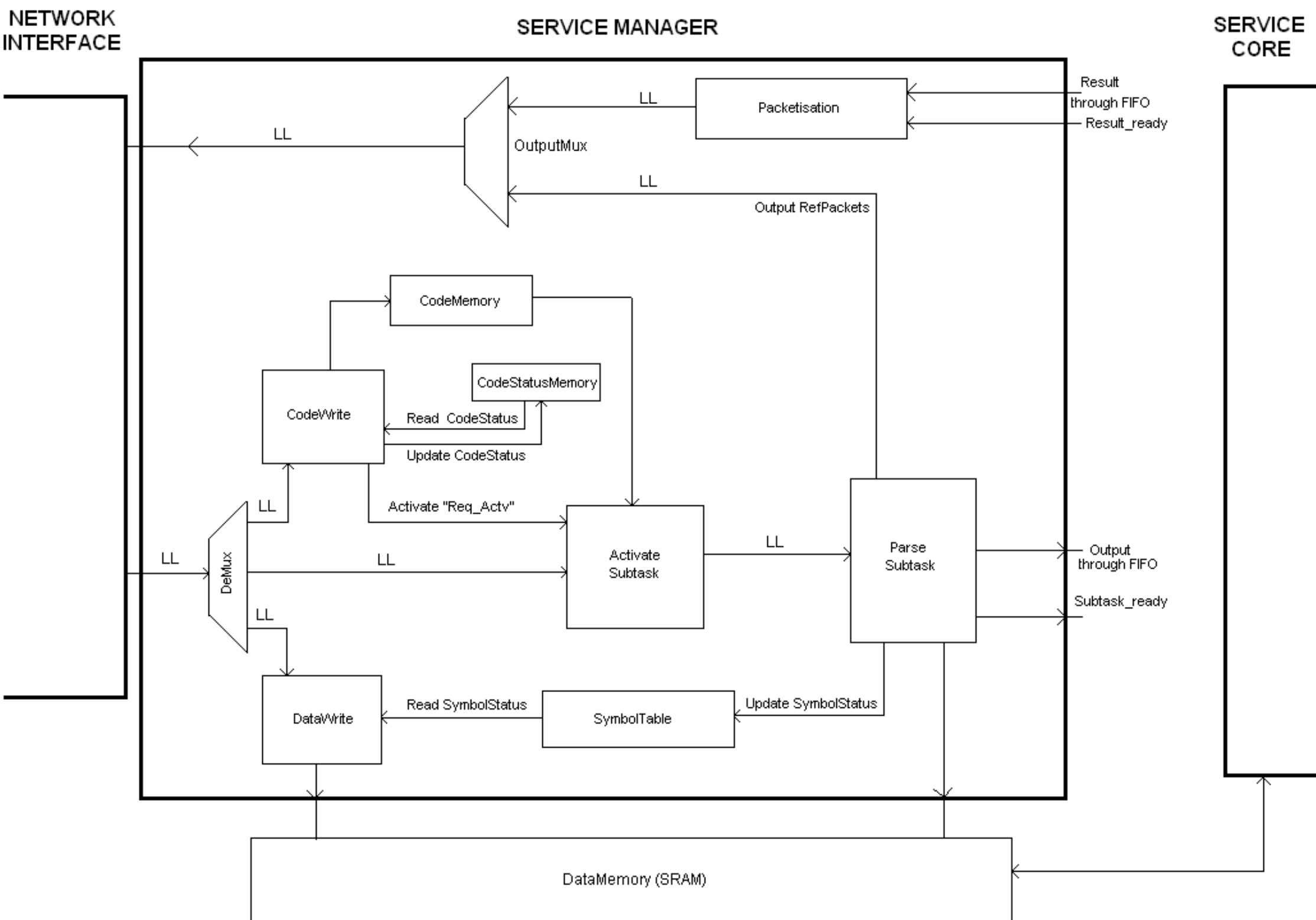
■ Architecture





Service Manager Implementation

- Present implementation provides the minimal functionality to process the three basic packets types:
 1. When a **data** packet enters the Service Manager \Rightarrow store in Data Memory
 2. When a **code** packet is received \Rightarrow store instruction code in the Code Memory.
 3. When a **reference** packet is received \Rightarrow activated corresp. code
 4. When code is activated \Rightarrow
 - a) create new reference packets for the respective **reference** symbols in the subtask; transmit reference packets onto NoC.
 - b) store **constant** symbols in the Data Memory.





Synthesis

- Target: Virtex-II Pro XC2VP30, speed grade -7 (XUP-V2P)
- Toolchain Xilinx ISE 9.2, XST

Optimisation Goal	Area	Speed
Slices	822 (6%)	917 (6%)
BRAMs	14 (10%)	8 (5%)
Max. clock speed	170MHz	242MHz



Simulations

- Inject a number of packets in the system
- All packets have 3 header symbols (1 word each)
- Code packet: $R0 \Rightarrow (S \ R1 \ R2 \ C1 \ R3 \ R4)$
- 6 instruction symbols (1 opcode + 5 operands)
 - 1 words each for S, R1..R4
 - 3 words for C1 (extended symbol)
- Reference packet: payload = ref to R0
- 2 data packets (payload 8 words)

New ReferencePacket created for every
RefSymbol and transmitted back over the network

The diagram illustrates a timing relationship between a clock signal and three data signals. The clock signal is a periodic square wave. The data signals are 32-bit buses. The top signal has a value of 32'h600098EE. The middle signal has a value of 32'h00000000. The bottom signal has a value of 32'h00D001E0. A zoomed-in view of the bottom signal shows its internal structure with 32-bit words and their corresponding addresses.

Address	Value
32'h00D001E0	32'hXXXXXXXX
32'h80D015EB	32'hXXXXXXXX
32'h80D014EA	32'hXXXXXXXX
32'hCCD00102	32'hXXXXXXXX
32'h80D012E8	32'hXXXXXXXX
32'h80D011E7	32'hXXXXXXXX
32'h80D010E6	32'hXXXXXXXX
32'h80D009E5	32'hXXXXXXXX



Simulations – summary

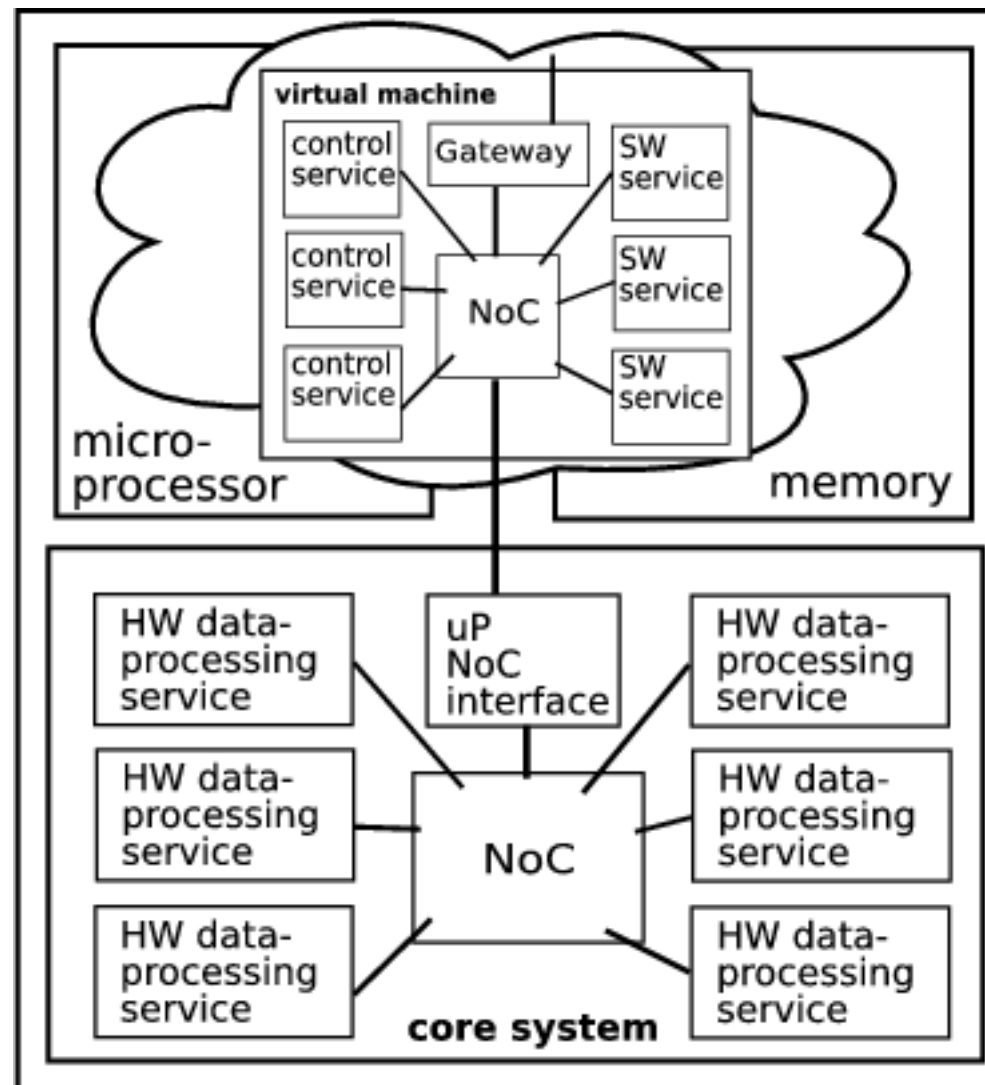
- Complete cycle takes 36 clock cycles
 - from start of receipt of code packet
 - to end of transmission of last reference packet
- 20 clock cycles + 4 clock cycles for each transmitted reference packet (1 cycle per word)



Comparison with software implementation



- Complete Gannet system: software + hardware





Speed comparison

- Gannet VM (C++) on an embedded PowerPC 405 processor running at 400MHz on a Virtex-II Pro FPGA
- Cycles required for task activation

	Cycle count
Service Manager	36
VM	250000



Area comparison

- Compared to the MicroBlaze soft processor core

	Slices	BRAMS	RAM (kB)
Service Manager	1504	24	512
MicroBlaze	917	8	20



Power consumption comparison

- Compared to PowerPC 405 core

	nJ/clock cyle	nJ/task activation	
Service Manager	1.03	30	
PowerPC	0.9	10^6	



Conclusion

- Introduced the Gannet service-based SoC architecture:
 - platform for task-level configuration of heterogeneous multi-core SoCs.
- Presented
 - architecture of the system
 - design and implementation of the Service Manager circuit
 - synthesis and simulation results for Xilinx Virtex-II Pro FPGA.
- Service Manager circuit meets the expectations:
 - small (less than 1000 slices)
 - low latency (30 clock cycles)
 - low power consumption (1nJ/clock cycle)

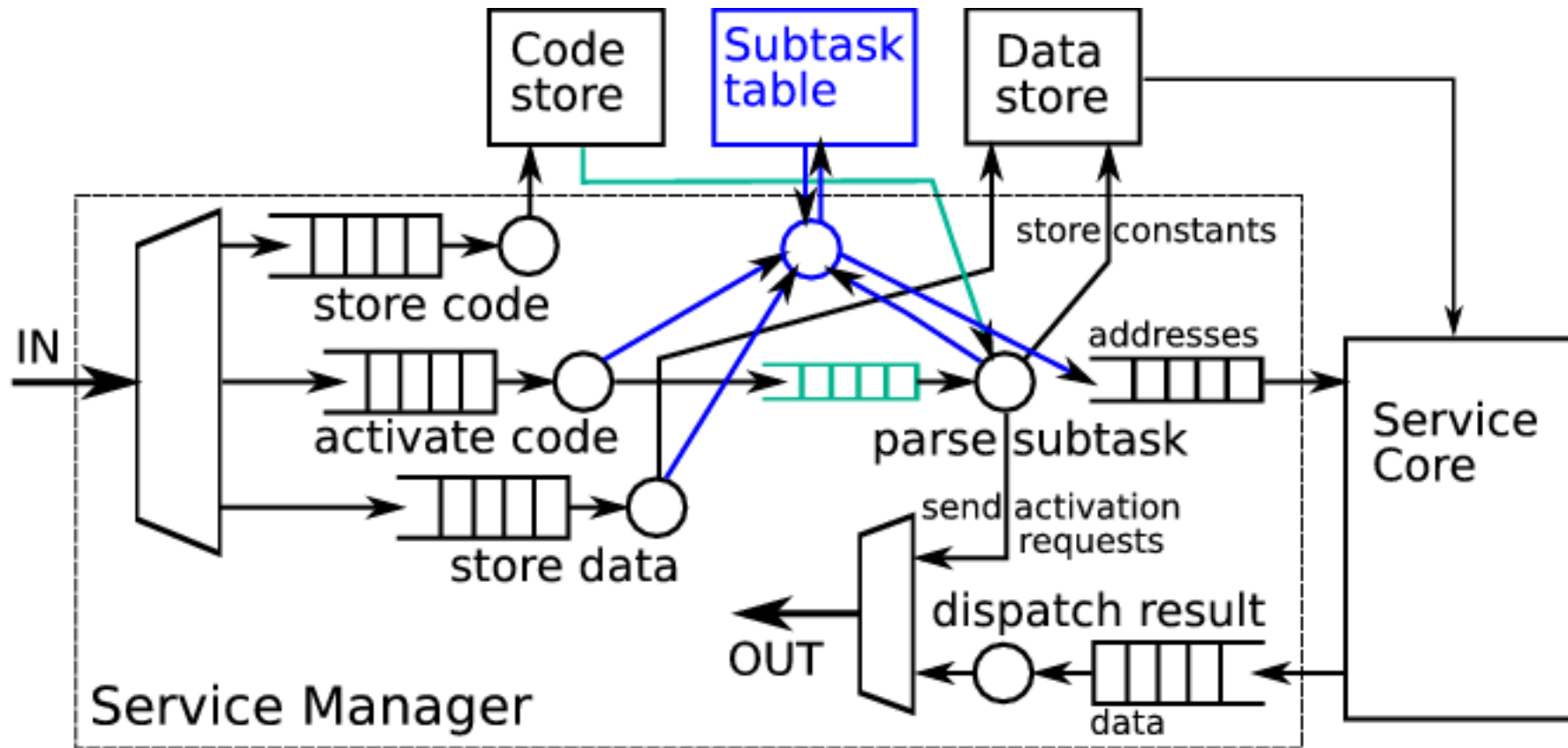


Future Work

- Performance
 - Streaming data processing
 - Data-driven operation
 - Caching and buffering
- Integration with NoC and microcontroller



Actual Design





Status

