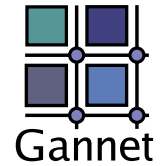




UNIVERSITY
of
GLASGOW



Gannet: a functional task description language for a service-based SoC architecture

Wim Vanderbauwhede
Department of Computing Science
University of Glasgow, UK



Overview

- What is a Gannet service-based System-on-Chip?
- How does it work? Gannet task descriptions
- The need for language constructs
- Language services by example
- Grammar and operational semantics
- Conclusion



What is Gannet?

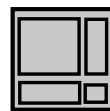
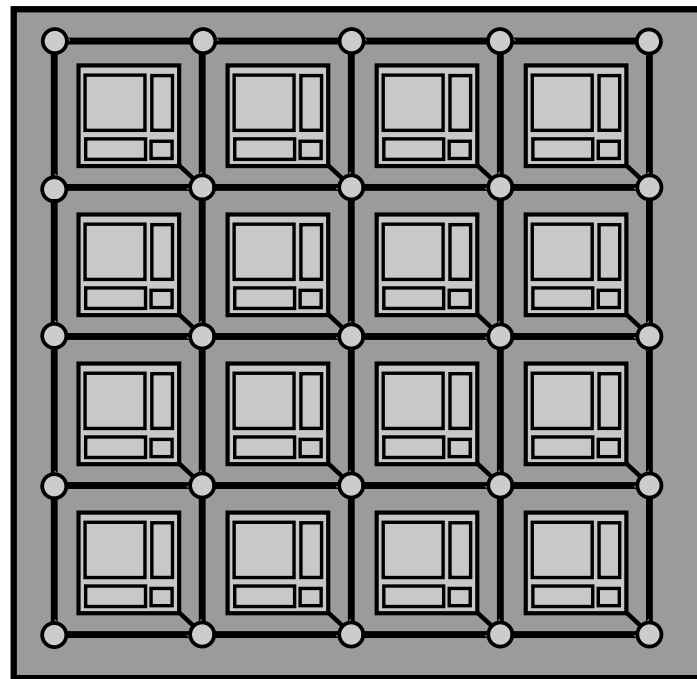
A distributed System-on-Chip (SoC) architecture

- a collection of processing cores (HW/SW)
- each core offers a a specific **service**
- all services are **fully connected** over an on-chip network (NoC)
- all information is transferred as **packets** over the NoC

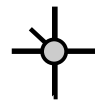


What is Gannet?

A distributed System-on-Chip (SoC) architecture



tile with
IP core



NoC switch



What is Gannet?

A functional machine language

- conceptually: an intermediate language, comparable to assembly language.
- syntactically and semantically: a pure functional language, similar to Scheme.
- the Gannet SoC architecture can be considered as a computing platform, a "machine" to run the Gannet language.

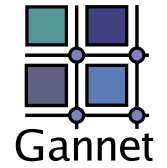


Why Gannet?

- tomorrow's SoC's will be **very big** (10^{10} logic gates)
 - traditional bus-style interconnect causes a bottleneck:
 - Synchronisation over large distances is impossible
 - Fixed point-to-point result in huge wire overhead
 - **on-chip networks** provide a solution
 - globally asynchronous/locally synchronous
 - flexible connectivity
- design reuse is essential => IP ("Intellectual Property") cores
- IP cores are highly complex, self-contained units
- treating such blocks as **services** is a logical abstraction



How does a Gannet SoC work?



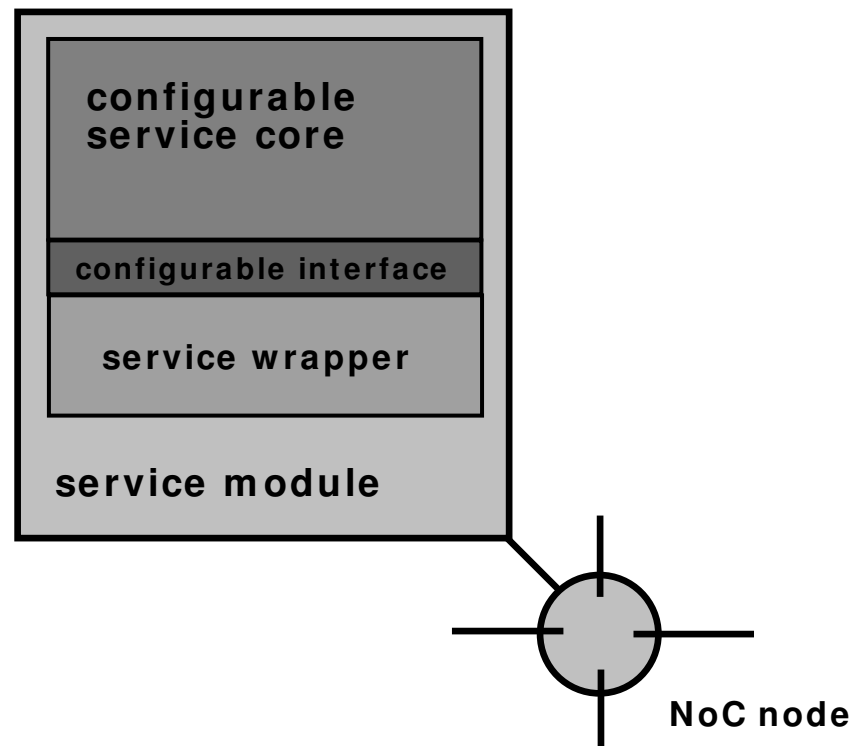
The services collaborate in a demand-driven dataflow fashion:

- **data** enter the system
- to be processed by **services**
- the **results** of which are, like the data, processed by services
- this process evolves according to a predefined but configurable **task**
- the **description** of such a task is a Gannet **program**



Managing the services

- to manage the flow of **data** and **task descriptions** between the **heterogenous service cores**, every core interfaces with the system through a **service manager**





Gannet's service manager

- the task description is a list of **symbols** representing either **data** or **services**
- essentially, the service manager uses two rules to evaluate the task description:
 - Data \Rightarrow request
 - Service \Rightarrow delegate
- it does the bookkeeping of all pending subtasks and the status of each of their arguments
- the service cores are task-agnostic



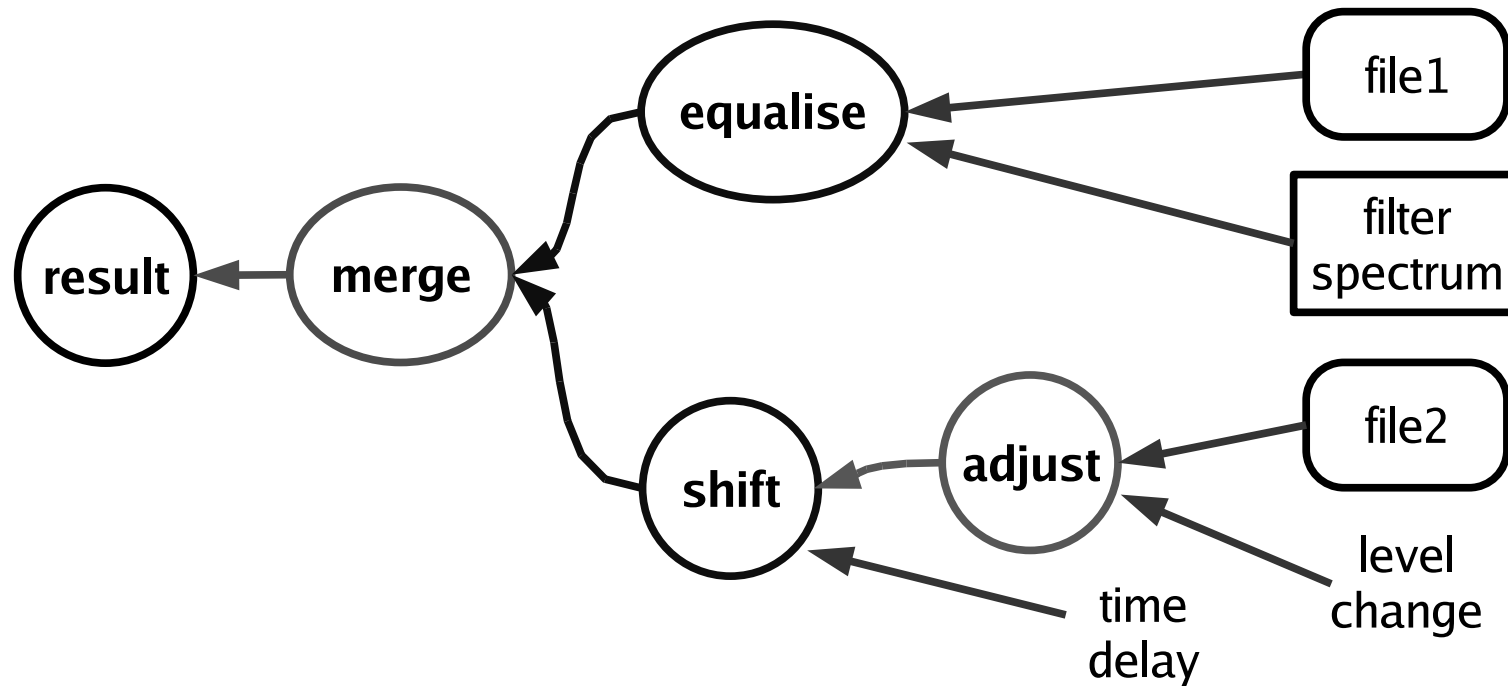
Gannet Task Description

- Suppose the system merges a number of audio files after having applied some processing to each file. E.g.:
 - Apply filtering with a given spectrum to the first file
 - Change the level of the second file and shift the wave a few seconds to synchronise with the first file
 - Merge both files



Gannet Task Description

Example task: a system to process audio files.





Syntax

- Human-readable task description format: S-expressions
- EBNF:
 - $expression ::= (service_symbol argument_symbol+)$
 - $argument_symbol ::= expression | data_symbol$

Example:

```
(merge  
  (shift time_delay  
    (adjust level_change file2))  
  (equalise filter_spectrum file1))
```



Symbols

- Symbol has 5 fields: Kind, Task, Subtask, Name, Count
- In the current prototype, a symbol is 64 bits long

Example in symbols:

```
S 1 9 merge          9
S 1 5 shift          5
D 1 1 time_delay      1
S 1 3 adjust        3
D 1 1 level_change    1
...
```



The Gannet task description approach

- allows in principle to describe arbitrary complex tasks
- but has severe limitations:
 - memory requirements
 - limited parallelism – no fan-in
 - no conditional branching
 - no loop constructs – program size

Solution: Adding language services



- Lexically scoped variables reduce memory utilisation

(**group**

(**assign** 'v1 (**group**

(**assign** 'v1 (S d11 d12))

(**assign** 'v2 (S d21 d22))

(S v1 v2))

(**assign** 'v2 (S d41 d42))

(S v1 v2))

- The `assign` service binds the result of a service call.
- The `group` service is creates the scope.



The Gannet Quote

- The quote ' indicates to the service manager that the following argument symbols must not be requested but **passed on as-is** to the service core
- Is implemented as a separate symbol

(S1 'd1)

Symbols:

S 1 1 S1 3

Q 1 1 ' 2

D 1 1 d1 1



Parallelism

- Service cores can produce multiple results.
- But a service can only return a single value.

List values: **list** service

(**group**

(**assign** 'l1 (S1 d1 d2))

(**assign** 'l2 (**list**

(S2 (**head** l1))

(S3 (**head** (**tail** l1)))

(**concat** l1 (list d4)))

(S4 (**head** l1) (**head** l2)))



Conditional branching

- Depending on some condition, subtasks might be delegated to different services.

Conditional branching: **if** service

```
(if (S_test_condition d1) (S1 d1) (S2 d1))
```

```
(if (S_test_condition d1) ' (S1 d1) ' (S2 d1))
```

- Quoting causes lazy evaluation, but is optional



Program size

- Without iterative or recursive constructs and subroutines or functions, a task description could become very large

Lambda functions and recursion: **lambda** and **apply** services

```
(group  
  (assign 'f (lambda 'x 'y' (* x y)))  
  (apply f d1 d2)  
)
```



Operational semantics

- For a Gannet language program in the context of the Gannet machine
- Independent on the syntax introduced above
- Context-sensitive reduction semantics (Felleisen)
- Evaluation of a service expression has two distinct, atomic stages:
 - **marshalling** stage (M)
 - **processing** stage (P)



Grammar

- Symbols and Symbol lists ($\langle \dots \rangle$: symbol list boundary)

$$\textit{symbol-list} ::= \langle (\textit{symbol} \mid \textit{symbol-list})^+ \rangle$$

- Expressions

$$\textit{expression} ::= \textit{variable-symbol}$$
$$\mid \textit{service-expression}$$
$$\mid \textit{quoted-expression}$$
$$\textit{service-expression} ::= \langle \textit{service-symbol} \textit{expression}^+ \rangle$$
$$\textit{quoted-expression} ::= \langle \textit{quote-symbol} \textit{expression}^+ \rangle$$
$$\mid \langle \textit{quote-symbol} \textit{argument-symbol} \rangle$$



- Language service expressions

$$\textit{group-expr} ::= \langle \mathbf{group} \textit{ assign-expr} + \textit{ expr} + \rangle$$
$$\textit{assign-expr} ::= \langle \mathbf{assign} \textit{ quote-symbol} \textit{ variable-symbol} \textit{ expr} \rangle$$
$$\textit{lambda-expr} ::= \langle \mathbf{lambda} \textit{ quoted-arg-list} \textit{ quoted-expr} \rangle$$
$$\textit{quoted-arg-list} ::= \langle \textit{ quote-symbol} \textit{ argument-symbol} \rangle *$$
$$\textit{apply-expr} ::= \langle \mathbf{apply} (\textit{ lambda-expr} | \textit{ variable-symbol}) \textit{ expr} + \rangle$$



Operational semantics

■ Shorthand notation

expression : e

quoted – expression : qe

service – expression : se

service – symbol : s

variable – symbol : v

argument – symbol : x

number – symbol : n

value : w



Operational semantics

- Evaluation context

$$C ::= [] \mid \langle s \dots C \dots \rangle$$

- Store

- The Gannet machine does not have global, shared memory; every service has its own local memory, with read-only access for the other services
- The **store()** concept must be contextualised (subscript to indicate context of store)



Non-language services

- Delegate Service

$$C[\langle s \dots se_1 \dots \rangle] \rightarrow^M C[s \dots w_1 \dots]$$

- Request Data

$$\begin{aligned} & (store_{data}(\dots (d_1 w_1) \dots) C[\langle s \dots d_1 \dots \rangle]) \\ & \rightarrow^M (store_{data}(\dots (d_1 w_1) \dots) C[s \dots w_1 \dots]) \end{aligned}$$

- Store Quoted expression

$$C[\langle s \dots qe_1 \dots \rangle] \rightarrow^M C[s \dots e_1 \dots]$$



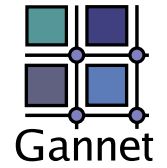
Language services

■ Variables

$$\begin{aligned} & (\mathbf{store}_{\text{assign}}(\dots) C[\langle \mathbf{assign} \mathit{qv} \mathit{e} \rangle]) \\ \rightarrow^M & (\mathbf{store}_{\text{assign}}(\dots) C[\langle \mathbf{assign} \mathit{v} \mathit{w} \rangle]) \\ \rightarrow^P & (\mathbf{store}_{\text{assign}}(\dots(\mathit{v} \mathit{w})\dots) C[\#t]) \end{aligned}$$

■ Grouping

$$\begin{aligned} & (\mathbf{store}_{\text{assign}}(\dots) C[\langle \mathbf{group} \dots \langle \mathbf{assign} \mathit{qv}_i \mathit{e}_i \rangle \dots \mathit{e} \rangle]) \\ \rightarrow^M & (\mathbf{store}_{\text{assign}}(\dots(\mathit{v}_i \mathit{w}_i)\dots) C[\langle \mathbf{group} \dots \#t \dots \mathit{w} \rangle]) \\ \rightarrow^P & (\mathbf{store}_{\text{assign}}(\dots) C[\mathit{w}]) \end{aligned}$$



- Function definition and application

$$\begin{aligned}
 & C[\langle \mathbf{lambda} \mathit{qx}_1 \dots \mathit{qx}_i \dots \mathit{qx}_n \mathit{qe}_a \rangle] \\
 & \rightarrow^M C[\langle \mathbf{lambda} x_1 \dots x_i \dots x_n e_a \rangle] \\
 & \rightarrow^P C[\langle x_1 \dots x_i \dots x_n e_a \rangle]
 \end{aligned}$$

- Function definition and application

$$\begin{aligned}
 & (\mathbf{store}_{\mathbf{apply}}(\dots) C[\langle \mathbf{apply} \langle \mathbf{lambda} \mathit{qx}_1 \dots \mathit{qx}_i \dots \mathit{qx}_n \mathit{qe}_a \rangle e_1 \dots e_i \dots e_n \rangle]) \\
 & \rightarrow^M (\mathbf{store}_{\mathbf{apply}}(\dots) C[\langle \mathbf{apply} \langle x_1 \dots x_i \dots x_n e_a \rangle w_1 \dots w_i \dots w_n \rangle]) \\
 & \rightarrow^P (\mathbf{store}_{\mathbf{apply}}(\dots (x_1 w_1) \dots (x_i w_i) \dots (x_n w_n)) C[e_a[x_i/w_i]])
 \end{aligned}$$



Conclusion

- Gannet project: facilitate high abstraction-level design of complex SoCs
- Novel **service-based** SoC architecture
- Distributed processing system
- Executes tasks defined in terms of cooperating services
- Functional **task description** language.
- Introducing **language services** to improve system performance.
- Next step: optimising the language to minimise resource utilisation