Tydi-lang: A Language for Typed Streaming Hardware

Zaid Al-Ars, Yongding Tian, Matthijs A. Reukers, Peter Hofstee, Matthijs Brobbel, Johan Peltenburg and Jeroen van Straten

Accelerated Big Data Systems group

Email: z.al-ars@tudelft.nl, Web: abs.ewi.tudelft.nl

Delft University of Technology

Overview

- Addressing HW design limitations
- Details of the Tydi specification and type-oriented streaming protocol
- Tydi-lang toolchain
- High-level abstraction of Tydi-lang
- SQL to Tydi-lang cases

Designing FPGA accelerators is complex

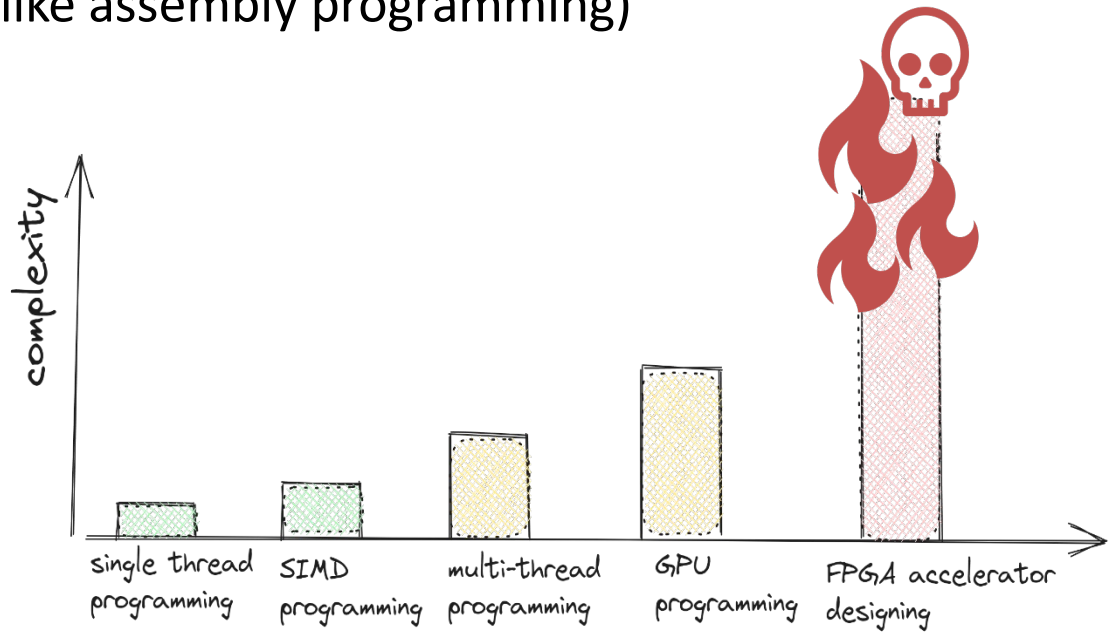
- Lack of data abstractions
- Low-level attributes (like assembly programming)

=> Large codebase

=> not composable

=> complex to debug

=> etc.



Adrian Sampson, https://www.sigarch.org/hdl-to-adl/

Data interfacing and communication challenge

- HW kernels design is not the only challenge
- Data interfacing and communication is a bigger challenge
  - Alignment of bus bandwidth
  - Communication synchrosization
  - Debugging bit-wise signals rather than variables
  - Etc.

Tydi specification to facilitate streaming of complex data

- **Tydi** is open specification to abstract streaming data in HW
- Automates HW design of streaming data interfaces
- Allows HW components (aka Streamlets) to be composed together
- It provides the following:
  - Data types
  - Data organization
  - Interface requirements

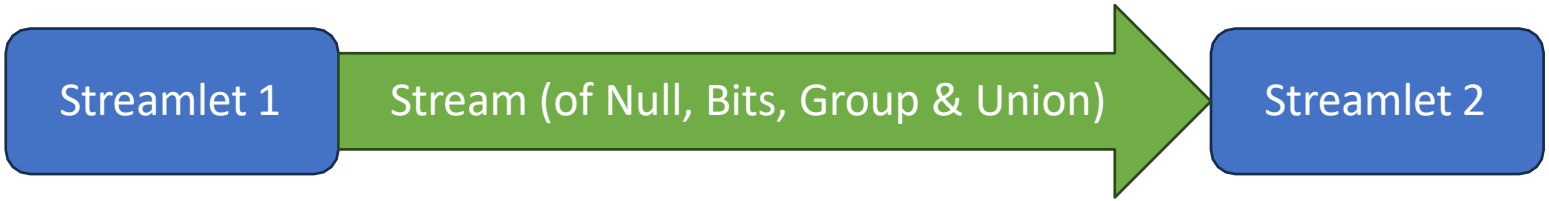
Peltenburg et al., Tydi: an open specification for complex data structures over hardware streams, IEEE Micro, 2020

Tydi specification to facilitate streaming of complex data: Data types

Tydi provides a type system for composite and variable-length data

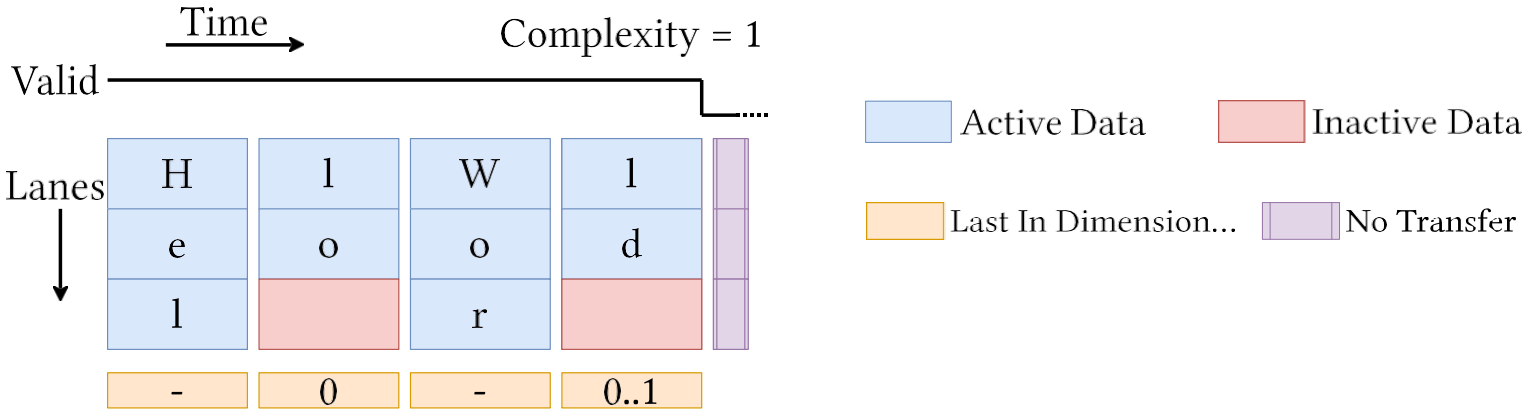
Type system defines the following data types

1. Stream: represents physical stream carrying the following logical types
2. Bits(N): represents a data signal of N bits
3. Group: composites of multiple types (all types set at the same time)
4. Union: composites of multiple types (one type can be active at a time)
5. Null: user-defined data type



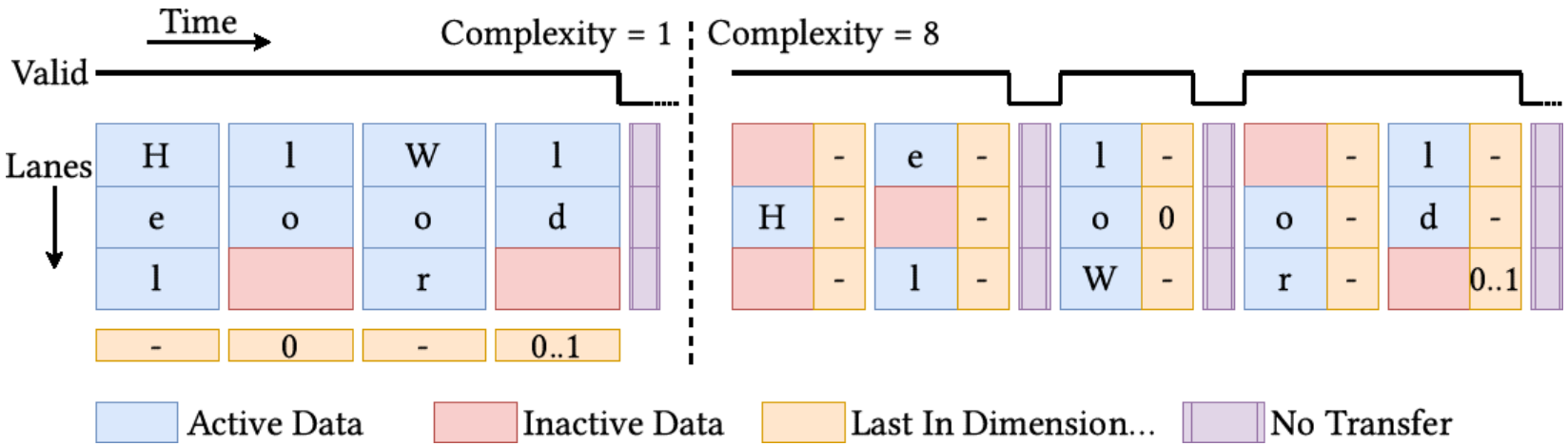
Tydi specification to facilitate streaming of complex data: Data organization

- **Tydi** defines how data elements are organized in transfers
- Nested data types: dimensionality property indicates if data is part of a sequence
- Translated to a “last” signal in HW
- Higher dimensionality need multiple last signals for nested sequences



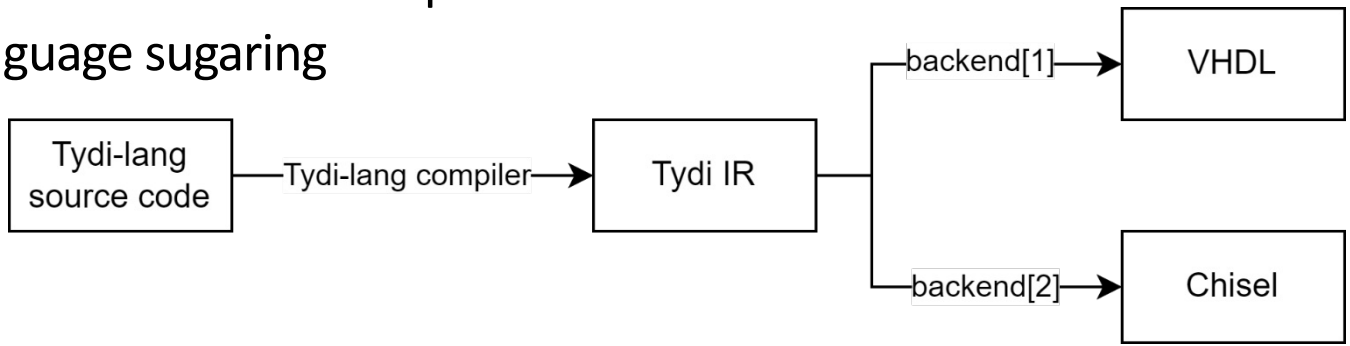
Tydi specification to facilitate streaming of complex data: Requirements

- **Tydi** defines the requirements system needs from transfers
- Tydi provides the following requirements attributes: Throughput, Direction, Synchronicity, Complexity



Tydi-lang: a language for typed streaming HW

- Tydi-lang is a domain-specific HDL based on **Tydi** specification
- Syntax inspired by Python and Rust
- Language features:
  - Hardware description by variables and types
  - Abstract hardware templates
  - Language sugaring



[1] M. Reukers, “A toolchain for streaming dataflow accelerator designs for big data analytics: Defining an ir for composable typed streaming dataflow designs, Fourteenth International Workshop on Accelerating Analytics and Data Management Systems Using Modern Processor and Storage Architectures, 2023
[2] Casper Cromjongh, “Enabling Collaborative and Interface-Driven Data-Streaming Accelerators Design with Tydi-Chisel”, IEEE Nordic Circuits and Systems Conference, 2023

Tydi-lang: HW description by variables & types

Declare const values (int, float, string, bool, time domain) and Tydi types (Bit, Group, Union, Stream, Null):

```
const year_max = 10^5 - 1;
type year_t = Bit(ceil(log2(year_max)));
type year_stream = Stream(year_t);  //a stream to represent values 0~9999
……
type Group Date {
  year: year_t,
  month: month_t,
  day: day_t,
};  //combine these bit streams
```

```
streamlet birthday_check_s {
  birth_date: date_stream in,
  pass: bool_stream out,
  ..
};  //use Tydi types to describe components
```

Tydi-lang: HW description with component templates

- Template allows designers to describe abstract components with generic Tydi types and variables.
- **Example 1:** void_s is designed to acknowledge all Tydi handshake signals regardless of the data. It is used when ports are not connected.

```
streamlet void_s<type_in: type> {    input: type_in in,    };
```

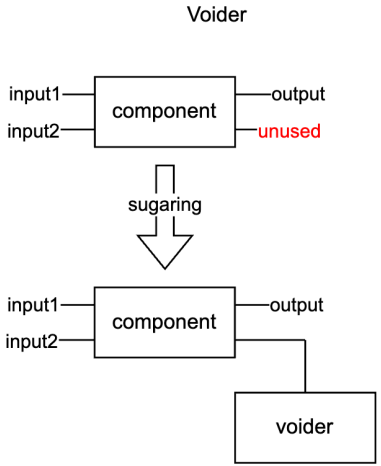
- **Example 2:** We can also define a duplicator to duplicate streaming packets. Useful when a value is accessed multiple times in programming logic.

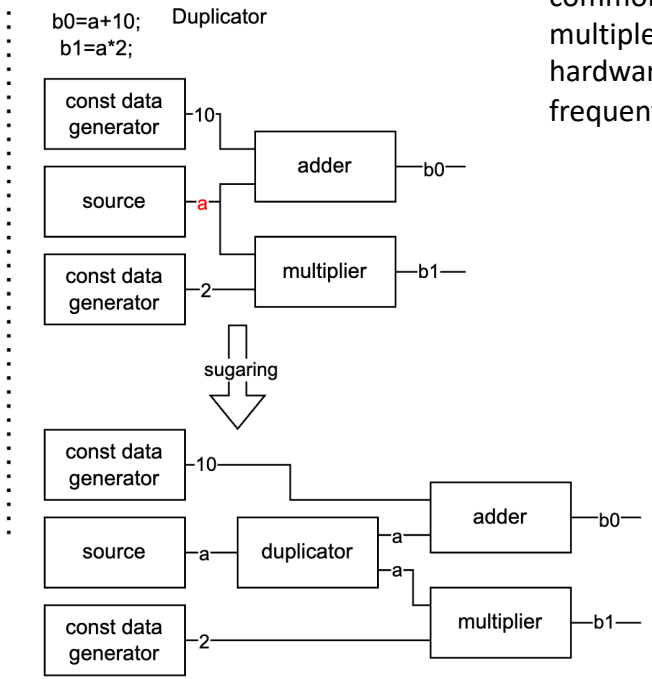
```
streamlet duplicator_s<data_type: type, output_channel: int> {
  input: data_type in,
  output: data_type [output_channel] out,
};
```

Tydi-lang: HW sugaring

Automatic insertion of *void_s* and *duplicator_s*

There should be a mechanism to handle unused ports, otherwise they would be blocked by the handshaking protocol.





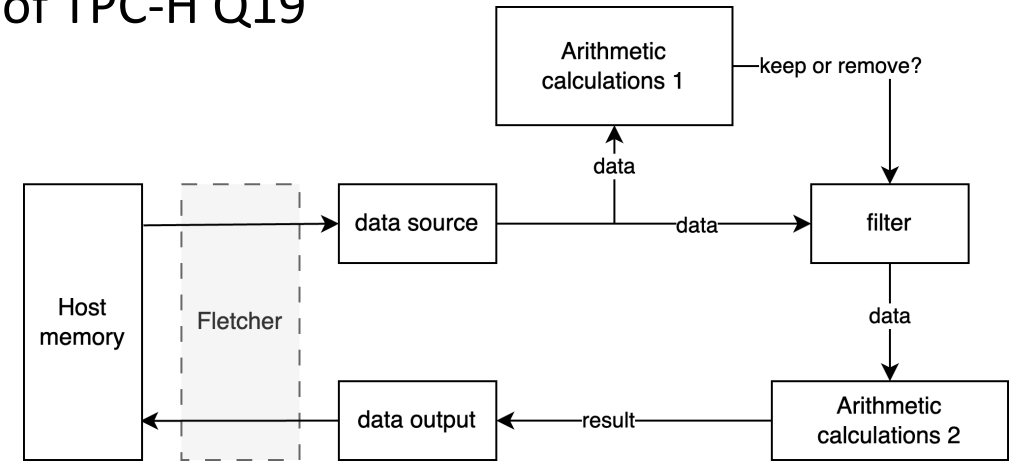
In software programming, it is common for a value to be used multiple times. Similarly in hardware design, we need to frequently duplicate streams.

Tydi-lang: SQL to Tydi-lang example

Translating SQL to Tydi-lang, we use TPC-H query 19 as an example:

```
select                                                    Arithmetic calculations 2
                sum(l_extendedprice* (1 - l_discount)) as
revenue
from                                                      Data source
                lineitem,
                part
where
                (                                         Arithmetic calculations 1
                                p_partkey = l_partkey
                                and p_brand = ':1'
                                and p_container in ('SM
CASE', 'SM BOX', 'SM PACK', 'SM PKG')
                                and l_quantity >= :4 and
l_quantity <= :4 + 10
                                and p_size between 1 and 5
                                and l_shipmode in ('AIR',
'AIR REG')
                                and l_shipinstruct =
'DELIVER IN PERSON'
                )
                …
```

Block diagram showing streamlets of TPC-H Q19

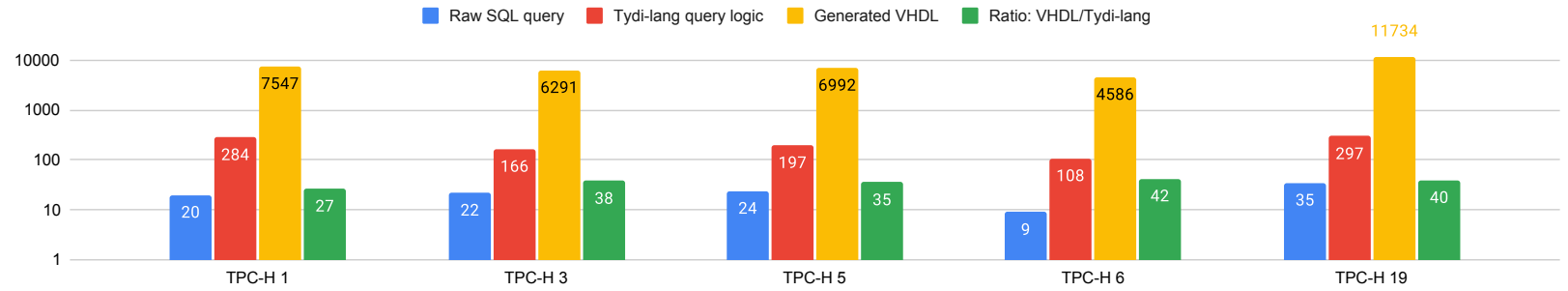


Tydi-lang: SQL to Tydi-lang example

We implement several TPC-H queries in Tydi-lang and compare the #lines of code for SQL/Tydi-lang/VHDL.

The Tydi-lang code contains three parts:

1) The Fletcher part, generated by Fletcher to access in-memory Arrow data (LoC = 166)
2) The Tydi-lang standard template library, including some frequently-used component templates (LoC = 151)
3) Tydi-lang code to performing query logic (LoC shown below)



Conclusions

1. Tydi spec: standard to describe composite & complex data

2. We present Tydi-lang: language based on Tydi spec, allowing higher abstractions of streaming components

3. We implement the logic of several TPC-H queries with Tydi-lang. Find that Tydi-lang can save over 20x LoC compared to writing VHDL directly

4. Future work: 1) emitting Chisel, & 2) automatic behavior code generation for templates