Ways to Constrain them: Generic Package

- Step 2: Create a „generic Package“:

```
use work.my_package.all;

package my_generic_package is
  generic (
    data_bits     : positive;
    user_bits     : positive := 1;
  );

  subtype sized_record is my_record(
    signal_3(data_bits - 1 downto 0),
    signal_4(user_bits - 1 downto 0)
  );
end package;
```

-> Include package with record

-> „generic Package“

-> Add generics as needed

-> Default values are also allowed

-> Create a subtype of the record and constrain it with package-generics

Ways to Constrain them: Generic Package

- How to use it:

```
package my_record_d32_u3 is
  new work.my_generic_package
    generic map (
      user_bits    => 3,
      data_bits    => 32
    );
use work.my_record_d32_u3.all;
```

Create new package with needed generics,

and include it

```
                .
                .
                .
architecture rtl of my_entity is

  signal my_signal    : sized_record;
```

Then just use the already constrained record

Ways to Constrain them: Generic Package

- Need more then one of these records in one entity?

```
architecture rtl of my_entity is
  package my_record_d32_u3 is
    new work.my_generic_package
      generic map (
        user_bits    => 3,
        data_bits    => 32
      );
  package my_record_d16_u1 is
    new work.my_generic_package
      generic map (
        user_bits    => 1,
        data_bits    => 16
      );
  signal my_signal_d32_u3    : my_record_d32_u3.sized_record;
  signal my_signal_d16_u1    : my_record_d16_u1.sized_record;
```

Create packages inside architecture

Call sized record from needed package

Ways to Constrain them: Propagation

- Example:
- Constraining is also automaticaly possible through propagation
- Unconstrained record can be used directly as a port, without constraining it
- Constraining will be provided through connected signal of the higher hierarchy
- => Constraining propagation is happening top-down

Ways to Constrain them: Propagation

- Example:

```
use work.my_package.all;

entity my_toplevel is
port (
);
end entity;

architecture rtl of my_toplevel is
signal my_signal : my_record(signal_3(7 downto 0), signal_4(0 to 2));
begin

instance : entity work.my_entity
port map(
    Data_in  => my_signal,
    Data_out => my_signal
);
```

Assign constraint signals

```
use work.my_package.all;

entity my_entity is
port (
    Data_in  : in  my_record;
    Data_out : out my_record
);
end entity;

architecture rtl of my_entity is
```

Example: AXI4Stream

- The Record Package:

```
library ieee;
use     ieee.std_logic_1164.all;


package axi4stream is
→type t_axi4stream_m2s is record
→→valid : std_logic;
→→data  : std_logic_vector;
→→last  : std_logic;
→end record;
→
→type t_axi4stream_s2m is record
→→ready : std_logic;
→end record;

→type t_axi4stream_m2s_vector is array(natural range <>) of t_axi4stream_m2s;
→type t_axi4stream_s2m_vector is array(natural range <>) of t_axi4stream_s2m;
end package;
```

Create one record in direktion master-to-slave (m2s)
-> record is unconstrained

Create one record in direktion slave-to-master (s2m)

Example: AXI4Stream

- Why do we need to separate records?

- inout is not working for most synthesis tools
- Assigning signals is only possible in one direction
- Is confusing to get the direction of each element
- => Best practice to separate to a M2S and S2M record

Example: AXI4Stream

- The Demultiplexer Entity

```
entity AXI4Stream_DeMux is
  port (
    Clock               : in  std_logic;
    Reset               : in  std_logic;
    -- Control interface
    DemuxControl        : in  std_logic_vector;
    -- IN Port
    In_M2S              : in  T_AXI4STREAM_M2S;
    In_S2M              : out T_AXI4STREAM_S2M;
    -- OUT Ports
    Out_M2S             : out T_AXI4STREAM_M2S_VECTOR;
    Out_S2M             : in  T_AXI4STREAM_S2M_VECTOR
  );
end entity;
```

One-Hot-Encoded

One Input Stream

To

N Output Streams

Example: AXI4Stream

- The Demultiplexer architecture

```
architecture rtl of AXI4Stream_Mux is
  signal Ready_v : std_logic_vector(Out_M2S'range);
begin
  assert DemuxControl'length = Out_M2S'length
    report "AXI4Stream_DeMux: Size of 'DemuxControl' is unequal to size of Stream-Vector"
    severity FAILURE;

  mapping_gen : for i in Out_M2S'range generate
    Out_M2S(i).data  <= In_M2S.data;
    Out_M2S(i).last  <= In_M2S.last;

    Out_M2S(i).valid <= In_M2S.valid and DemuxControl(i);
    Ready_v(i)       <= Out_S2M(i).ready and DemuxControl(i);
  end generate;

  In_S2M.ready <= or(Ready_v);
end architecture;
```

Create signal with same range as Port-Vector

Check for Sizes

Generate through all Ports
Data related signals can be directly connected

Propagate Ready and Valid if Controlbit is set

Set Ready if one enabled Channel is Ready

VHDL-2019 outlook

- Introduces „Mode Views“

```
view my_record_MasterView of my_record is
    signal_1    : in;
    signal_2    : out;
end view my_record_MasterView;
```

- Opposite with new attribute „converse“

```
view my_record_SlaveView is my_record_MasterView'converse;
```

VHDL-2019 outlook

- Introduces „Mode Views“

```
view my_record_MasterView of my_record is
    signal_1    : in;
    signal_2    : out;
end view my_record_MasterView;
```

- Opposite with new attribute „converse“

```
view my_record_SlaveView is my_record_MasterView'converse;
```

- Within port declaration

```
entity my_entity is
    port (
        Master : view my_record_MasterView;
        Slave  : view (my_record_SlaveView) of my_record_V(0 to 1)
    );
```