Specification Refinement
Refinement

- Refinement is used to reflect the condition after the partitioning and the interface between HW/SW is built
  - **Refinement** is the update of specification to reflect the mapping of variables.

- Functional objects are grouped and mapped to system components
  - Functional objects: variables, behaviors, and channels
  - System components: memories, chips or processors, and buses

- Specification refinement is very important
  - Makes specification consistent
  - Enables simulation of specification
  - Generate input for synthesis, compilation and verification tools
Refining variable groups

- The memory to which the group of variables are reflected and refined in specification.

- Variable folding:
  - Implementing each variable in a memory with a fixed word size

- Memory address translation
  - Assignment of addresses to each variable in group
  - Update references to variable by accesses to memory
Variable folding

```plaintext
variable A : bit_vector(3 downto 0);
variable B : bit_vector(15 downto 0);
variable C : bit_vector(11 downto 0);
variable D : bit_vector(11 downto 0);
```

8-bit Memory

![Diagram showing variable folding](image-url)
Memory address translation

**Original specification**

```plaintext
variable J, K : integer := 0;
variable V : IntArray (63 downto 0);

V(K) := 3;
X := V(36);
V(J) := X;

for J in 0 to 63 loop
  SUM := SUM + V(J);
end loop;
```

**Refined specification**

```plaintext
variable J, K : integer := 0;
variable MEM : IntArray (255 downto 0);

MEM(K + 100) := 3;
X := MEM(136);
MEM(J+100) := X;

for J in 100 to 163 loop
  SUM := SUM + MEM(J);
end loop;
```

**Assigning addresses to V**

- V (63 downto 0)
- MEM(163 downto 100)

**Refined specification without offsets for index J**

```plaintext
variable J : integer := 100;
variable K : integer := 0;
variable MEM : IntArray (255 downto 0);

MEM(K + 100) := 3;
X := MEM(136);
MEM(J) := X;

for J in 100 to 163 loop
  SUM := SUM + MEM(J);
end loop;
```
Channel refinement

- Channels: virtual entities over which messages are transferred
- Bus: physical medium that implements groups of channels
- Bus consists of:
  - wires representing data and control lines
  - protocol defining sequence of assignments to data and control lines
- Two refinement tasks
  - Bus generation: determining bus width
    - number of data lines
  - Protocol generation: specifying mechanism of transfer over bus
Communication

- Shared-memory communication model
  - Persistent shared medium
  - Non-persistent shared medium

- Message-passing communication model
  - Channel
    - uni-directional
    - bi-directional
    - Point-to-point
    - Multi-way
  - Blocking
  - Non-blocking

- Standard interface scheme
  - Memory-mapped, serial port, parallel port, self-timed, synchronous, blocking
Inter-process communication paradigms:
(a) shared memory, (b) message passing
Characterizing communication channels

- For a given behavior that sends data over channel C,
  - **Message size** $\text{bits}(C)$
    - number of bits in each message
  - **Accesses**: $\text{accesses}(B,C)$
    - number of times $P$ transfers data over $C$
  - **Average rate** $\text{averate}(C)$
    - rate of data transfer of $C$ over lifetime of behavior
  - **Peak rate** $\text{peakrate}(C)$
    - rate of transfer of single message

\[
\text{bits}(C) = 8 \text{ bits} \\
\text{averate}(C) = \frac{24 \text{ bits}}{400 \text{ ns}} = 60 \text{ Mbits/s} \\
\text{peakrate}(C) = \frac{8 \text{ bits}}{100 \text{ ns}} = 80 \text{ Mbits/s}
\]
Characterizing buses

- For a given bus B
  - **Buswidth** \( \text{buswidth}(B) \)
    - number of data lines in B
  - **Protocol delay** \( \text{protdelay}(B) \)
    - delay for single message transfer over bus
  - **Average rate** \( \text{averate}(B) \)
    - rate of data transfer over lifetime of system
  - **Peak rate** \( \text{peakrate}(B) \)
    - maximum rate of transfer of data on bus

\[
\text{peakrate}(C) = \frac{\text{buswidth}(B)}{\text{portdelay}(B)}
\]
Determining bus rates

- Idle slots of a channel used for messages of other channels
- To ensure that channel average rates are unaffected by bus
  \[ \text{average}(B) = \sum_{C \in B} \text{average}(C) \]
- Goal: to synthesize a bus that constantly transfers data for channel
  \[ \text{peakrate}(B) = \text{average}(C) \]
Constraints for bus generation

- **Bus-width**: affects number of pins on chip boundaries
- **Channel average rates**: affects execution time of behaviors
- **Channel peak rates**: affects time required for single message transfer

![Diagram illustrating constraints and rates](image-url)
Bus generation algorithm

- **Compute buswidth range**: minwidth=1, maxwidth = Max(bit(C))
- **For** minwidth: currwidth ≤ maxwidth **loop**
  - **Compute bus peak rate**:
    \[ \text{peakrate}(B) = \frac{\text{currwidth}}{\text{protdelay}(B)} \]
  - **Compute channel average rates**
    \[ \text{commtime}(B) = \text{access}(B,C) \times \left[ \text{currwidth} \times \text{protdelay}(B) \right] \]
    \[ \text{average}(C) = \frac{\text{access}(B,C) \times \text{bits}(C)}{\text{comptime}(B) + \text{commtime}(B)} \]
  - **If** peakrate(B) ≥ Σ averate(C) **then**
    \[ \text{if } \text{bestcost} > \text{ComputeCost(currwidth)} \]
    \[ \text{bestcost} = \text{ComputeCost(currwidth)} \]
    \[ \text{bestwidth} = \text{currwidth} \]
Bus generation example

- Assume
  - 2 behavior accessing 16 bit data over two channels
  - Constraints specified for channel peak rates

<table>
<thead>
<tr>
<th>Channel C</th>
<th>Behavior B</th>
<th>Variable accessed</th>
<th>Bits(C)</th>
<th>Access(B, C)</th>
<th>Comptime(p)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CH1</td>
<td>P1</td>
<td>V1</td>
<td>16 data + 7 addr</td>
<td>128</td>
<td>515</td>
</tr>
<tr>
<td>CH2</td>
<td>P2</td>
<td>V2</td>
<td>16 data + 7 addr</td>
<td>128</td>
<td>129</td>
</tr>
</tbody>
</table>
Protocol generation

- Bus consists of several sets of wires:
  - **Data lines**, used for transferring message bits
  - **Control lines**, used for synchronization between behaviors
  - **ID lines**, used for identifying the channel active on the bus
- All channels mapped to bus share these lines
- Number of data lines determined by bus generation algorithm
- Protocol generation consists of five steps
Protocol generation steps

- 1. Protocol selection
  - full handshake, half-handshake etc.
- 2. ID assignment
  - N channels require log2(N) ID lines
Protocol generation steps

- 3 Bus structure and procedure definition
  - The structure of bus (the data, control, ID lines) is defined in the specification.

- 4. Update variable-reference
  - References to a variable that has been assigned to another component must be updated.

- 5. Generate processes for variables
  - Extra behavior should be created for those variables that have been sent across a channel.
Protocol generation example

type HandShakeBus is record
    START, DONE : bit;
    ID : bit_vector(1 downto 0);
    DATA : bit_vector(7 downto 0);
end record;

signal B : HandShakeBus;

procedure ReceiveCH0( rxdata : out bit_vector) is
begin
    for J in 1 to 2 loop
        wait until (B.START = '1') and (B.ID = "00")
        rxdata (8*J-1 downto 8*(J-1)) <= B.DATA;
        B.DONE <= '1';
    end loop;
end ReceiveCH0;

procedure SendCH0( txdata : in bit_vector) is
begin
    bus B.ID <= "00";
    for J in 1 to 2 loop
        B.data <= txdata(8*J-1 downto 8*(J-1));
        B.START <= '1';
        wait until (B.DONE = '1');
        B.START <= '0';
        wait until (B.DONE = '0');
    end loop;
end SendCH0;
Refined specification after protocol generation
Resolving access conflicts

- System partitioning may result in concurrent accesses to a resource
  - Channels mapped to a bus may attempt data transfer simultaneously
  - Variables mapped to a memory may be accessed by behaviors simultaneously

- Arbiter needs to be generated to resolve such access conflicts

- Three tasks
  - Arbitration model selection
  - Arbitration scheme selection
  - Arbiter generation
Arbitration models

STATIC

Dynamic
Arbitration schemes

- Arbitration schemes determines the priorities of the group of behaviors’ access to solve the access conflicts.

- Fixed-priority scheme statically assigns a priority to each behavior, and the relative priorities for all behaviors are not changed throughout the system’s lifetime.
  - Fixed priority can be also pre-emptive.
  - It may lead to higher mean waiting time.

- Dynamic-priority scheme determines the priority of a behavior at the run-time.
  - Round-robin
  - First-come-first-served
Refinement of incompatible interfaces

● Three situation may arise if we bind functional objects to standard components:
  ● Neither behavior is bound to a standard component.
    – Communication between two can be established by generating the bus and inserting the protocol into these objects.
  ● One behavior is bound to a standard component
    – The behavior that is not associated with standard component has to use dual protocol to the other behavior.
  ● Both behaviors are bound to standard components.
    – An interface process has to be inserted between the two standard components to make the communication compatible.
Effect of binding on interfaces
Protocol operations

- Protocols usually consist of five atomic operations
  - waiting for an event on input control line
  - assigning value to output control line
  - reading value from input data port
  - assigning value to output data port
  - waiting for fixed time interval

- Protocol operations may be specified in one of three ways
  - Finite state machines (FSMs)
  - Timing diagrams
  - Hardware description languages (HDLs)
Protocol specification: FSMs

- Protocol operations ordered by sequencing between states
- Constraints between events may be specified using timing arcs
- Conditional & repetitive event sequences require extra states, transitions

![Protocol Pa Diagram](image)

- $\text{ADDR}_p \leftarrow \text{AddrVar}(7 \text{ downto } 0)$; $\text{ARDY}_p \leftarrow '1'$
- $(\text{ARCV}_p='1')$
- $\text{ADDR}_p \leftarrow \text{AddrVar}(15 \text{ downto } 8)$; $\text{AREQ}_p \leftarrow '1'$
- $(\text{DRDY}_p='1')$
- DataVar $\leftarrow \text{DATAp}$

![Protocol Pb Diagram](image)

- $(\text{RD}_p='1')$
- $\text{MAaddrVar} \leftarrow \text{MADDR}_p$
- $(100\text{ns})$
- $\text{MDATAp} \leftarrow \text{MemVar}(\text{MAaddrVar})$
Protocol specification: Timing diagrams

- **Advantages:**
  - Ease of comprehension, representation of timing constraints

- **Disadvantages:**
  - Lack of action language, not simulatable
  - Difficult to specify conditional and repetitive event sequences
Protocol specification: HDLs

- **Advantages:**
  - Functionality can be verified by simulation
  - Easy to specify conditional and repetitive event sequences

- **Disadvantages:**
  - Cumbersome to represent timing constraints between events

```vhdl
port ADDRp : out
bit_vector(7 downto 0):
port DATAp : in
bit_vector(15 downto 0);
port ARDYp : out bit;
port ARCVp : in bit;
port DREQp : out bit;
port DRDYp : in bit;
ADDRp <= AddrVar(7 downto 0);
ARDYp <= '1';
wait until (ARCVp = '1');
ADDRp <= AddrVar(15 downto 8);
DREQp <= '1';
wait until (DRDYp = '1');
DataVar <= DATAp;
```

```vhdl
port MADDRp : in
bit_vector(15 downto 0);
port MDATAp : out
bit_vector(15 downto 0);
port RDp : in bit;
wait until (RDp = '1');
MAddrVar := MADDRp;
wait for 100 ns;
MDATAp <= MemVar(MAddrVar);
```
Interface process generation

- **Input:** HDL description of two fixed, but incompatible protocols
- **Output:** HDL process that translates one protocol to the other
  - i.e. responds to their control signals and sequence their data transfers
- **Four steps required for generating interface process (IP):**
  - Creating relations
  - Partitioning relations into groups
  - Generating interface process statements
  - Interconnect optimization
IP generation: creating relations

- Protocol represented as an ordered set of relations
- Relations are sequences of events/actions

**Protocol Pa**

```vhdl
ADDRp <= AddrVar(7 downto 0);
ARDYp <= '1';
wait until (ARCVp = '1');
ADDRp <= AddrVar(15 downto 8);
DREQp <= '1';
wait until (DRDYp = '1');
DataVar <= DATAp;
```

**Relations**

- **A1** [ (true) ]:
  ```vhdl
  ADDRp <= AddrVar(7 downto 0)
  ARDYp <= '1'
  ```
- **A2** [ (ARCVp = '1') ]:
  ```vhdl
  ADDRp <= AddrVar(15 downto 8)
  DREQp <= '1'
  ```
- **A3** [ (DRDYp = '1') ]:
  ```vhdl
  DataVar <= DATAp
  ```
IP generation: partitioning relations

- Partition the set of relations from both protocols into groups.
- Group represents a unit of data transfer

<table>
<thead>
<tr>
<th>Protocol Pa</th>
<th>Protocol Pb</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 (8 bits out)</td>
<td>B1 (16 bits in)</td>
</tr>
<tr>
<td>A2 (8 bits out)</td>
<td></td>
</tr>
<tr>
<td>A3 (16 bits in)</td>
<td>B2 (16 bits out)</td>
</tr>
</tbody>
</table>

G1=(A1 A2 B1)  G2=(B1 A3)
IP generation: inverting protocol operations

- For each operation in a group, add its dual to interface process
- Dual of an operation represents the complementary operation
- Temporary variable may be required to hold data values

<table>
<thead>
<tr>
<th>Atomic operation</th>
<th>Dual operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>wait until (Cp = '1')</td>
<td>Cp &lt;= '1'</td>
</tr>
<tr>
<td>Cp &lt;= '1'</td>
<td>wait until (Cp = '1')</td>
</tr>
<tr>
<td>var &lt;= Dp</td>
<td>Dp &lt;= TempVar</td>
</tr>
<tr>
<td>Dp &lt;= var</td>
<td>TempVar := Dp</td>
</tr>
<tr>
<td>wait for 100 ns</td>
<td>wait for 100 ns</td>
</tr>
</tbody>
</table>

Interface Process

/* (group G1)' */
wait until (ARDYp = '1');
TempVar1(7 downto 0) := ADDRp ;
ARCVp <= '1';
wait until (DREQp = '1');
TempVar1(15 downto 8) := ADDRp ;
RDp <= '1';
MADDRp <= TempVar1;

/* (group G2)' */
wait for 100 ns;
TempVar2 := MDATAp ;
DRDYp <= '1';
DATAp <= TempVar2 ;
IP generation: interconnect optimization

- Certain ports of both protocols may be directly connected
- Advantages:
  - Bypassing interface process reduces interconnect cost
  - Operations related to these ports can be eliminated from interface process
Transducer synthesis

- Input: Timing diagram description of two fixed protocols
- Output: Logic circuit description of transducer
- Steps for generating logic circuit from timing diagrams:
  - Create event graphs for both protocols
  - Connect graphs based on data dependencies or explicitly specified ordering
  - Add templates for each output node in combined graph
  - Merge and connect templates
  - Satisfy min/max timing constraints
  - Optimize skeletal circuit
Generating event graphs from timing diagrams

e.g. FIFO stack control cell
Deriving skeletal circuit from event graph

- **Advantages:**
  - Synthesizes logic for transducer circuit directly
  - Accounts for min/max timing constraints between events

- **Disadvantages:**
  - Cannot interface protocols with different data port sizes
  - Transducer not simulatable with timing diagram description of protocols
Hardware/Software interface refinement
Tasks of hardware/software interfacing

- Data access (e.g., behavior accessing variable) refinement
- Control access (e.g., behavior starting behavior) refinement
- Select bus to satisfy data transfer rate and reduce interfacing cost
- Interface software/hardware components to standard buses
- Schedule software behaviors to satisfy data input/output rate
- Distribute variables to reduce ASIC cost and satisfy performance
Summary

- Refinement of variable groups: variable folding, address translation
- Refinement of channel groups: bus and protocol generation
- Resolution of access conflicts: arbiter generation
- Refinement of incompatible interfaces: IP generation, transducer synthesis