Refinement

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

- **Refinement** is update of specification to reflect mapping

- Need for refinement
  - Makes specification consistent
  - Enables simulation of specification
  - Generate input for synthesis, compilation and verification tools
Outline

- Refining variable groups
- Channel refinement
- Resolving access conflicts
- Refining incompatible interfaces
Refining variable groups

- Group of variables mapped to a memory

- 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) ;

7 0
C(11 downto 8) D(11 downto 6) B(15 downto 8) C(7 downto 0) D(5 downto 0) B(7 downto 0)
```

8-bit Memory

- to variable C in memory
- to variable D in memory
Memory address translation

**Original specification**

```
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;
....
```

**Assigning addresses to V**

```
V (63 downto 0)
```

```
MEM(163 downto 100)
```

**Refined specification**

```
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 0 to 63 loop
   SUM := SUM + MEM(J+100);
end loop;
....
```

**Refined specification without offsets for index J**

```
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;
....
```
Refining channel groups

- Channels are virtual entities over which messages are transferred
- Bus is a 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 buswidth i.e. number of data lines
  - *Protocol generation*: specifying mechanism of transfer over bus
Characterizing communication channels

- For a given behavior $P$ that sends data over channel $C$
  - **Message size**, $\text{bits}(C')$ : number of bits in each message
  - **Accesses**, $\text{accesses}(P, C)$ : 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

```
channel X

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>X1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>X2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>X3</td>
<td></td>
</tr>
</tbody>
</table>
```

$t=0$  100  200  300  400

```
bits(C') = 8 bits
averate(C') = \frac{24 \text{bits}}{400 \text{ns}} = 60 \text{ Mbits/s}
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 $B$ over lifetime of system
  - **Peak rate**, $\text{peakrate}(B)$: maximum rate of transfer of data on bus

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

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

- **Buswidth**: 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
Bus generation algorithm [NG94]

/* Determine range of buswidths */
minwidth = 1, maxwidth = Max(bits(C))

mincost = ∞, mincostwidth = ∞
for currwidth in minwidth to maxwidth loop
  /* compute bus peak rate */
  peakrate(B) = currwidth ÷ protdelay(B)
  /* compute sum of channel average rates */
  averatesum = 0;
  for all channels C in B loop
    averate(C) = access(P, C) × bits(C)
    comptime(P) + commtime(P)
    averatesum = averatesum + averate(C);
  end loop
  if (peakrate(B) > averatesum) then
    /* feasible solution, determine minimal cost */
    currcost = ComputeCost(currwidth)
    if (currcost < mincost) then
      mincost = currcost, mincostwidth = currwidth
    end if
  end if
end loop
return(mincostwidth)
Bus generation algorithm

- Compute buswidth range: $\text{minwidth} = 1$, $\text{maxwidth} = \text{Max}(\text{bits}(C))$

- For $\text{minwidth} \leq \text{currwidth} \leq \text{maxwidth}$ loop
  - Compute bus peak rate:
    $$\text{peakrate}(B) = \frac{\text{currwidth}}{\text{protdelay}(B)}$$
  - Compute channel average rates
    $$\text{commt}(P) = \text{access}(P,C) \times \left[ \text{currwidth} \times \text{protdelay}(B) \right]$$
    $$\text{avrate}(C') = \frac{\text{access}(P,C') \times \text{bits}(C)}{\text{commt}(P) + \text{commt}(P)}$$
  - if $\text{peakrate}(B) \geq \sum_{C \in B} \text{avrate}(C')$ then
    - if $\text{bestcost} > \text{ComputeCost}($currwidth$)$ then
      $$\text{bestcost} = \text{ComputeCost}(\text{currwidth})$$
      $$\text{bestwidth} = \text{currwidth}$$
Bus generation example

- 2 behavior accessing 16 bit data over two channels
- Constraints specified for channel peak rates
Performance vs. buswidth tradeoffs

- Allows a buswidth to be selected, given performance constraints
  e.g. behavior P1 has performance constraint of 2500 clocks.
  buswidths of 4 or greater must be selected
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 six steps
Protocol generation

1. **Protocol selection**: full handshake, half-handshake etc.
2. **ID assignment**: \( N \) channels require \( \log_2(N) \) ID lines

```
behavior P
variable AD;
begin
  ....
  X <= 32 ;
  ....
  MEM(AD) := X + 7;
  ....
end ;

behavior Q
variable COUNT;
begin
  ....
  MEM(60) := COUNT ;
  ....
end ;
```

```
variable X : bit_vector(15 downto 0) ;

variable MEM : bit_vector (63 downto 0, 15 downto 0);

```

```
bus B

CH0

"00"

CH1

"00"

CH2

"00"

CH3

"00"

```
Protocol generation

3. Bus structure definition

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' ;
wait until (B.START = '0') ;
B.DONE <= '0' ;
end loop;
end ReceiveCH0;

4. Bus protocol definition

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;
Protocol generation

5. Update variable references
6. Generate behaviors for variables

process Q
variable COUNT;
begin
    ....
    SendCH3(60, COUNT);
    ....
    end;

process P
variable AD, Xtemp;
begin
    ....
    SendCH0(32);
    ....
    ReceiveCH1(Xtemp);
    SendCH2(AD, Xtemp+7);
    ....
    end;

process Xproc
variable X;
begin
    wait on B.ID;
    if (B.ID="00") then
        receiveCH0(X);
    elsif (B.ID="01") then
        sendCH1(X);
    end if;
    end;

process MEMproc
variable MEM: array(0 to 63);
begin
    wait on B.ID;
    if (B.ID="10") then
        receiveCH2(MEM);
    elsif (B.ID="11") then
        receiveCH3(MEM);
    end if;
    end;
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

Refinement
**Arbiter generation**

- Example of bus arbitration
  - Two behaviors accessing a single resource, bus $B$
  - Behavior $P$ assigned higher priority than $Q$
  - Fixed priority implemented with two handshake signals $Req$ and $Grant$

```vhdl
-- Arbiter generation

-- Process B_arbiter
begin
  wait until (Req_P='1') or (Req_Q='1');
  if (Req_P='1') then
    Grant_P <= '1';
    wait until (Req_P='0');
    Grant_P <= '0';
  elsif (Req_Q='1') then
    Grant_Q <= '1';
    wait until (Req_Q='0');
    Grant_Q <= '0';
  end if;
end process;

-- Process P
variable AD Xtemp;
begin
  Req_P <= '1';
  wait until (Grant_P='1');
  SendCH0(32);
  Req_P <= '0';
  end process;

-- Process Q
variable COUNT;
begin
  Req_Q <= '1';
  wait until (Grant_Q='1');
  SendCH3(60, COUNT);
  Req_Q <= '0';
  end process;

-- Process Xproc
variable X;
begin
  wait on B.ID;
  if (B.ID='00') then
    receiveCH0(X);
  elsif (B.ID='01') then
    sendCH1(X);
  end if;
end process;

-- Process MEMproc
variable MEM: array(0 to 63);
begin
  wait on B.ID;
  if (B.ID='10') then
    receiveCH2(MEM);
  elsif (B.ID='11') then
    receiveCH3(MEM);
  end if;
end process;
```
Effect of binding on interfaces

Custom

behavior A

protocol

Pa

Channel X

Custom

behavior B

protocol

Pb

Custom

behavior X

Pa

Channel X

Standard

Pb

behavior B

Standard

behavior A

Pa

Interface Process

Standard

Pb

behavior B
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

\[
\begin{align*}
& \text{start} \\
& a_1 \quad \text{ADDRp} \leq \text{AddrVar}(7 \text{ downto } 0); \\
& \quad \text{ARDYp} \leq \text{'}1\text{'}; \\
& (\text{ARCVp} = \text{'}1\text{'}) \\
& a_2 \quad \text{ADDRp} \leq \text{AddrVar}(15 \text{ downto } 8); \\
& \quad \text{AREQp} \leq \text{'}1\text{'}; \\
& (\text{DRDYp} = \text{'}1\text{'}) \\
& a_3 \quad \text{DataVar} \leq \text{DATAp}
\end{align*}
\]

Protocol Pb

\[
\begin{align*}
& \text{start} \\
& b_1 \quad (\text{RDp} = \text{'}1\text{'}) \\
& b_2 \quad \text{MAddrVar} := \text{MADDRp} \\
& (100 \text{ ns}) \\
& b_3 \quad \text{MDATAp} \leq \text{\text{MemVar} (MAddrVar)}
\end{align*}
\]
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 Pa
ARDYp
ADDRp
7..0 15..8
ARCVp
DREQp
DRDYp
DATAp
15..0

Protocol Pb
MADDRp
RDp
15..0
MDATAp
100ns
```
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

- 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) :
    ADDRp <= AddrVar(7 downto 0)
    ARDYp <= '1' ]

A2 [ (ARCVp = '1') :
    ADDRp <= AddrVar(15 downto 8)
    DREQp <= '1' ]

A3 [ (DRDYp = '1') :
    DataVar <= DATAp ]
IP generation: partitioning relations

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

\[ G_1 = (A_1 \ A_2 \ B_1) \quad G_2 = (B_1 \ A_3) \]
**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.

### Atomic operation vs. Dual operation

<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

```plaintext
/* (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 wait for 100 ns

wait for 100 ns;
TempVar2 := MDATAp;
DRDYp <= '1';
DATAp <= TempVar2;
```

**Diagram:**
- ADDRp
- DATAp
- ARDYp
- ARCVp
- DREQp
- DRDYp
- MADDRp
- MDATAp
- RDp
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

```
wait until (ARDYp = '1');
TempVar1(7 downto 0) := ADDRp;
ARDYp <= '1';
wait until (DREQp = '1');
TempVar1(15 downto 8) := ADDRp;
RDP <= '1';
MADDRp <= TempVar1;
wait for '100 ns;
DRDYp <= '1';
```

```
wait until (ARDYp = '1');
TempVar1(7 downto 0) := ADDRp;
ARDYp <= '1';
wait until (DREQp = '1');
TempVar1(15 downto 8) := ADDRp;
RDP <= '1';
MADDRp <= TempVar1;
wait for '100 ns;
DRDYp <= '1';
```
Transducer synthesis [BK87]

- 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

(a) Partitioned specification

(b) Mapping to architecture
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 and future directions

In this section, we described:
- 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

Future work should address the following issues:
- Effects of bus arbitration delays on performance of a behavior
- Developing metrics to guide selection of protocols and arbitration schemes
- Efficient synthesis of arbiter and interface processes