EENG 426/CPSC 459/ENAS 876
Silicon Compilation

Function block compilation

Computer Systems Lab

`http://csl.yale.edu/~rajit`

Fall 2018

Yale AVLSI

Function evaluation

How do we implement the following CHP?

$*[\ L?x;\ \ R!f(x)\ ]$

- pick representation for variables
- process decomposition
- parallel composition of parts

Yale AVLSI

Function block decomposition

First step: control data decomposition

$*[\ L?x;\ \ R!f(x)\ ]$

$\triangleright$

$*[\ L';R'\ ]$
$\|\ *[\ L' \bullet L?x\ ]$
$\|\ *[\ R' \bullet R!f(x)\ ]$

How do we implement:

$*[\ R' \bullet R!f(x)\ ]$

Yale AVLSI

Function block decomposition

Another alternative is to design the computation of the function as a "filter" on the output of channel R.
Normal handshake on channel R:

$*[\ R\Uparrow;\ \ [ri];\ \ R\Downarrow;\ \ [\neg ri]\ ]$

$R\Uparrow$: concurrent assignment of rails of R to a valid value
$R\Downarrow$: concurrent assignment of rails of R to a neutral value
Environment:

$*[\ [v(R)];\ \ ri\uparrow;\ \ [n(R)];\ \ ri\downarrow\ ]$

$v(R)$: true when the rails of R encode a valid data value
$n(R)$: true when the rails of R encode a neutral data value

Yale AVLSI

Function block decomposition

We modify this as follows:

$*[\ R' \Uparrow;\ [ri];\ R' \Downarrow;\ [\neg ri]\ ]$
$\|\ *[\ [v(R')];\ R \Uparrow;\ [n(R')];\ R \Downarrow\ ]$

where the data sent on R is a function of the data transmitted on R'.

The value is computed "on the rails" while the communication action is being performed.

- R becomes valid only after R' is valid
- R becomes neutral only after R' is neutral

Yale AVLSI

Delay-insensitive (DI) codes

We have been using dual rail codes for communication.

Codes where data is exchanged in a four phase protocol are delay insensitive when:

- we have valid and neutral values (disjoint)
- when bits change from a neutral to a valid value no intermediate value is neutral or valid
- when bits change from a valid to a neutral value no intermediate value is neutral or valid

What are valid and neutral values for dual rail codes?

(Note: valid/neutral values can be state-dependent)

Yale AVLSI

DI codes

Operations on rails (wires):

$$\{n(X)\} X \Uparrow \{v(X)\}$$

$$\{v(X)\} X \Downarrow \{n(X)\}$$

In each statement, there can be at most one assignment to each data rail used to encode values.

Some practical considerations:

- implementation of waits should be simple
- codes should be simple
- small number of wires

Yale AVLSI

DI codes

Distributive codes:

- X can be divided into sets of *subcodes*
- validity/neutrality can be defined for subcodes
- If S is the set of subcodes, then
  - $(\forall y : y \in S : n(Y)) \Rightarrow n(X)$
  - $(\forall y : y \in S : v(Y)) \Rightarrow v(X)$

Dual rail codes are distributive in the obvious way.

Yale AVLSI

DI codes

Berger codes:

- Two parts: data bits and check bits.
- Initially data bits zero, check bits zero (neutral value)
- Check bits: # number of zeros in data
- Valid value: check bits consistent

Yale AVLSI

DI codes

k-out-of-N codes:

- Initially all data bits are zero (neutral value)
- Valid value: k of the N bits are one

Some observations:

- One hot codes: $k = 1$
- Dual rail codes: $k = 1$, $N = 2$
- Any *subset* of codewords can be used as well

Sperner codes: $k = N/2$

Yale AVLSI

Function block decomposition

We focus on the implementation of:

$$*[\ [v(X)];\ Y\Uparrow;\ [n(X)];\ Y\Downarrow\ ]$$

When the code is distributive, we can apply the following transformations:

$$*[\ [v(X)]; Y\Uparrow; [n(X)]; Y\Downarrow\ ]$$

$\triangleright$

$$*[\ [(\wedge k :: v(X_k))]; Y\Uparrow; [(\wedge k :: n(X_k))]; Y\Downarrow\ ]$$

$\triangleright$

$$*[\ [(\wedge k :: v(X_k))]; (\| k :: Y_k\Uparrow); [(\wedge k :: n(X_k))]; (\| k :: Y_k\Downarrow)\ ]$$

Yale AVLSI

Function block decomposition

$$*[\ [(\wedge k :: v(X_k))]; (\| k :: Y_k\Uparrow); [(\wedge k :: n(X_k))]; (\| k :: Y_k\Downarrow)\ ]$$

$\triangleright$

$$*[\ (\| k :: [v(X_k)]; Y_k\Uparrow); (\| k :: [n(X_k)]; Y_k\Downarrow)\ ]$$

$\triangleright$

$$(\| k :: *[\ [v(X_k)]; Y_k\Uparrow; [n(X_k)]; Y_k\Downarrow\ ])$$

Why are these transformations valid?

Yale AVLSI

Function block examples

Example: complement

- $X_k = (xt_k, xf_k)$
- $Y_k = (yt_k, yf_k)$

$$
\begin{array}{l}
*[[xt_k \lor xf_k];[xt_k \longrightarrow yf_k\uparrow \;[\!]\; xf_k \longrightarrow yt_k\uparrow];\\
\quad [\neg xt_k \land \neg xf_k]; yt_k\downarrow, yf_k\downarrow\\
\;]
\end{array}
$$

Production rules:

$$
\begin{array}{rcl@{\qquad\qquad}rcl}
xt_k & \mapsto & yf_k\uparrow & xf_k & \mapsto & yt_k\uparrow\\
\neg xt_k & \mapsto & yf_k\downarrow & \neg xf_k & \mapsto & yt_k\downarrow
\end{array}
$$

Yale AVLSI

Function block examples

Example: logical AND

- $X_k = (at_k, af_k, bt_k, bf_k)$
- $Y_k = (yt_k, yf_k)$

$$
\begin{array}{l}
*[\;\;[(at_k \lor af_k) \land (bt_k \lor bf_k)];\\
\quad [at_k \land bt_k \longrightarrow yt_k\uparrow \;[\!]\; af_k \lor bf_k \longrightarrow yf_k\uparrow];\\
\quad [\neg at_k \land \neg af_k \land \neg bt_k \land \neg bf_k]; yt_k\downarrow, yf_k\downarrow\\
\;]
\end{array}
$$

Production rules:

$$
\begin{array}{rcl@{\qquad\qquad}rcl}
at_k \land bt_k & \mapsto & yt_k\uparrow & at_k \land bf_k \lor af_k \land (bt_k \lor bf_k) & \mapsto & yf_k\uparrow\\
\neg at_k \land \neg bt_k & \mapsto & yt_k\downarrow & \neg at_k \land \neg af_k \land \neg bt_k \land \neg bf_k & \mapsto & yf_k\downarrow
\end{array}
$$

Yale AVLSI

An adder function block

Design of a simple N-bit ripple-carry adder:

- Three inputs: a, b, and c
- Two outputs: s, d

We can use the function block compilation strategy, where we pretend for the moment that the carry-in for each bit of the adder is an input to the block.

Yale AVLSI

An adder function block

$$
\begin{array}{l}
*[[(at \lor af) \land (bt \lor bf) \land (ct \lor cf)];\\
\quad [at \land bt \lor (a \neq b) \land ct \longrightarrow dt\uparrow\\
\quad [\!]\, af \land bf \lor (a \neq b) \land cf \longrightarrow df\uparrow\\
\quad],\\
\quad [ct \land (a = b) \lor cf \land (a \neq b) \longrightarrow st\uparrow\\
\quad [\!]\, cf \land (a = b) \lor ct \land (a \neq b) \longrightarrow sf\uparrow\\
\quad];\\
\quad [\neg at \land \neg af \land \neg bt \land \neg bf \land \neg ct \land \neg cf];\\
\quad dt\downarrow, df\downarrow, st\downarrow, sf\downarrow\\
\;]
\end{array}
$$

Connect so that the d output of one stage is the c input for the next stage.

Yale AVLSI

Distribute the validity test

$$
\begin{array}{l}
*[\,[at \wedge bt \vee (a \neq b) \wedge ct \longrightarrow dt\uparrow \\
\quad [\!]\, af \wedge bf \vee (a \neq b) \wedge cf \longrightarrow df\uparrow \\
\quad], \\
\quad ([(at \vee af) \wedge (bt \vee bf) \wedge (ct \vee cf)]; \\
\quad [ct \wedge (a = b) \vee cf \wedge (a \neq b) \longrightarrow st\uparrow \\
\quad [\!]\, cf \wedge (a = b) \vee ct \wedge (a \neq b) \longrightarrow sf\uparrow \\
\quad]); \\
\quad [\neg at \wedge \neg af \wedge \neg bt \wedge \neg bf \wedge \neg ct \wedge \neg cf]; \\
\quad dt\downarrow, df\downarrow, st\downarrow, sf\downarrow \\
]
\end{array}
$$

The sum for bit position $k+1$ waits for the carry-out of bit position k to be valid!

Yale

AVLSI

Distribute the neutrality test

$$
\begin{array}{l}
*[\,[at \wedge bt \vee (a \neq b) \wedge ct \longrightarrow dt\uparrow \\
\quad [\!]\, af \wedge bf \vee (a \neq b) \wedge cf \longrightarrow df\uparrow \\
\quad], \\
\quad ([(at \vee af) \wedge (bt \vee bf) \wedge (ct \vee cf)]; \\
\quad [ct \wedge (a = b) \vee cf \wedge (a \neq b) \longrightarrow st\uparrow \\
\quad [\!]\, cf \wedge (a = b) \vee ct \wedge (a \neq b) \longrightarrow sf\uparrow \\
\quad]); \\
\quad [\neg at \wedge \neg af \wedge \neg bt \wedge \neg bf \longrightarrow dt\downarrow, df\downarrow], \\
\quad [\neg ct \wedge \neg cf \longrightarrow st\downarrow, sf\downarrow] \\
]
\end{array}
$$

The sum for bit position $k+1$ waits for the carry-out of bit position k to be neutral!

Yale

AVLSI

Adder: fblock production rules

Production rules:

$$
\begin{array}{rcl}
at \wedge bt \vee (af \wedge bt \vee at \wedge bf) \wedge ct & \mapsto & dt\uparrow \\
\neg at \wedge \neg af \wedge \neg bt \wedge \neg bf & \mapsto & dt\downarrow
\end{array}
$$

$$
\begin{array}{rcl}
af \wedge bf \vee (af \wedge bt \vee at \wedge bf) \wedge cf & \mapsto & df\uparrow \\
\neg at \wedge \neg af \wedge \neg bt \wedge \neg bf & \mapsto & df\downarrow
\end{array}
$$

$$
\begin{array}{rcl}
ct \wedge (at \wedge bt \vee af \wedge bf) \vee cf \wedge (at \wedge bf \vee af \wedge bt) & \mapsto & st\uparrow \\
\neg ct \wedge \neg cf & \mapsto & st\downarrow
\end{array}
$$

$$
\begin{array}{rcl}
cf \wedge (at \wedge bt \vee af \wedge bf) \vee ct \wedge (at \wedge bf \vee af \wedge bt) & \mapsto & sf\uparrow \\
\neg ct \wedge \neg cf & \mapsto & sf\downarrow
\end{array}
$$

Yale

AVLSI