In the following diagram, we present a circuit which implements an arbiter.

Using the SMV Input Language for Description of Circuits
The circuit accepts inputs \( r_{e0}, \ldots, r_{eN} \) and responds through outputs \( a_{ck0}, \ldots, a_{ckN} \). The latch (memory element) \( L \) represents a token which travels around the circuit in a cyclic mode. The latch is on at cell which has been persistently requested since the last visit of the token. At any point in time, if there is a cell which is currently requested and in which both \( M \) and \( L \) are on, then the request with the lowest index will be acknowledged. If no such cell exists, then the request with the lowest index will be acknowledged.

The purpose of Circuit

A. Pnueli
A Single Cell

In the following diagram we present the interface of the arbiter cell with its environment. We observe that a cell has input signals $req$, $ti$, $oi$, and $gi$ and outputs $to$, $oo$, $go$, and $ack$.

In the next slide we present how cells $C[0]$, $C[1]$, ..., $C[N-1]$ are interconnected to form the full arbiter circuit.
Interconnection Scheme

\[ \text{Interconnection Scheme} \]
How the Circuit Operates

Each cell \( C[i] \) reads the input signal \( \text{req} \) and may respond by an output to signal \( \text{ack} \).

The chain of signals produced at every step from \( C[i] \) to \( C[j] \) transmits the token.

The chain of \( \text{gi} \) signals produces a grant signal, such that \( C[i] \cdot \text{gi} = 0 \) if for some \( j \) or \( \text{req} \).

The chain of \( \text{go} \) signals produces an override signal, such that \( C[i] \cdot \text{go} = 1 \) if for some \( j \) or \( \text{req} \).

The chain of \( \text{ack} \) signals for hardware verification, NYU, Fall, 2004.
A First Version of File arbiter.smv

VAR

MODULE main

DEFINEN:=4;N-1:=N-1;

VAR

C[0]:Cell(1,C[N].to,C[1].oo,!C[0].oo);
C[1]:Cell(0,C[0].to,C[2].oo,C[0].go);
C[2]:Cell(0,C[1].to,C[3].oo,C[1].go);
C[3]:Cell(0,C[2].to,C[4].oo,C[2].go);
C[N]:Cell(0,C[N-1].to,0,C[N-1].go);

ASSIGN

oo : boolean! oo : boolean!
to : boolean! to : boolean!
go : boolean! go : boolean!
ack : boolean! ack : boolean!
reg : boolean! reg : boolean!
rW : boolean! rW : boolean!
rT : boolean! rT : boolean!
req : boolean! req : boolean!

JUSTICE

MODULE Cell(t_init,ti,oi,gi)

VAR

ASSIGN

init(rW):=0;init(rT):=t_init;
next(rW):=req&(rW|rT);
next(rT):=ti;
ack:=req&(gi|rT&rW);
to:=rT;oo:=oi|rT&rW;go:=gi&!req;

MODULE arbiter

VAR

A.Pnueli
A more efficient SMV representation of the circuit assigns SMV variables only to latches and input variables. Such representation is presented in the next slide.

Note that this script file checks properties of representative cells, such as exclusion between $C[2]$ and $C[3]$, and eventual response for $C[2]$.

```
File arbiter2.pf
Print "Check properties of Arbiter"
Print "Check Mutual Exclusion"
Let exclusion := !(C[2].ack & C[3].ack);
Call Invariance (exclusion);
Print "Check Eventual response"
Call Temp_Entail (C[2].req, !C[2].req | C[2].ack);
Print "Absence of unsolicited response"
Call Invariance (C[2].ack -> C[2].req);
```

A more efficient SMV representation of the circuit assigns SMV variables only to latches and input variables. Such representation is presented in the next slide.
This version uses `DEFINE` instead of variables declaration, and uses loop constructors for array of definitions.

This version uses `DEFINE` instead of variables declaration, and uses loop constructors for array of definitions.

An Improved Version of File arbiter.smv

```smv
MODULE main

DEFINEN:=4;N-1:=N-1;

VAR

for(i=1;i<=N-1;i=i+1)

{C[i]:Cell(0,C[i-1].to,C[i+1].oo,C[i-1].go);}

C[N]:Cell(0,C[N-1].to,0,C[N-1].go);

MODULE Cell(T_init,ti,oi,gi)

VAR

rW:boolean;rT:boolean;req:boolean;

DEFINE

ack:=req&(gi|rT&rW);
to:=rT;oo:=oi|rT&rW;go:=gi&!req;

ASSIGN

init(rW):=0;init(rT):=T_init;

next(rW):=req&(rW|rT);next(rT):=ti;

JUSTICE1

This version uses `DEFINE` instead of variables declaration, and uses loop constructors for array of definitions.

VAR

reg : boolean; req : boolean!

MODULE Cell1(T_init,tt,tt',oo',tt'')

(C[N-1].to,C[N].to)

{ (C[T].to,C[T-1].to,C[T+1].to,C[T].oo,C[T-1].oo,C[T+1].oo,C[T].i1,C[T].i0)

fo) FOR (T = 1 TO N-1)

(VAR

N-1 =: N-1

DEFINE

main
```

Hardware Verification, NYU, Fall, 2004
Lecture 3: Using SMV for Hardware Verification, NYU, Fall, 2004

An Improved Script File

The following script file checks properties of all cells.

```verbatim
AnImprovedScriptFile

A. Pnueli

CallTempEntail(\text{C}[\text{t}].\text{req} \equiv \text{C}[\text{t}].\text{req})

CallInvvariance(exclusion)

Let exclusion = exclusion \& !\text{C}[\text{t}].\text{ack} \& !\text{C}[\text{t}].\text{ack}

For(t \in 0 \ldots N)

Print "\text{Check properties of Arbiter}"\n
CallInvvariance(\text{C}[\text{t}].\text{ack} \rightarrow \text{C}[\text{t}].\text{req})

For(i \in 0 \ldots N)

For(j \in i+1 \ldots N)

Let exclusion := exclusion \& !!(\text{C}[i].\text{ack} \& !\text{C}[j].\text{ack})

End--For(j \in i+1 \ldots N)

End--For(i \in 0 \ldots N)

CallInvvariance(exclusion)

End -- For(t \in 0 \ldots N)

End -- For(t \in 0 \ldots N)

End -- check_all

End -- check_all
```

Hardware Verification, NYU, Fall, 2004

62