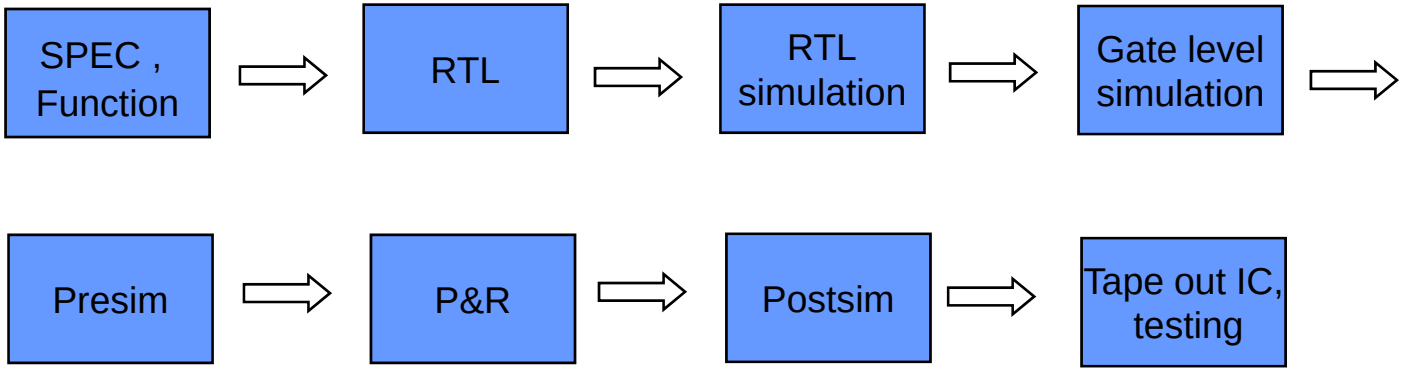
Network IC Design
Case Study: Ethernet MAC

Prof. Ying-Dar Lin
National Chiao Tung University
ydlin@cs.nctu.edu.tw

ASIC vs. FPGA

- **Usually we use a FPGA before tape out an ASIC**
- **FPGA get more delay than ASIC**
- **Design flow :**



- **Difference in Gate level simulation and P&R**

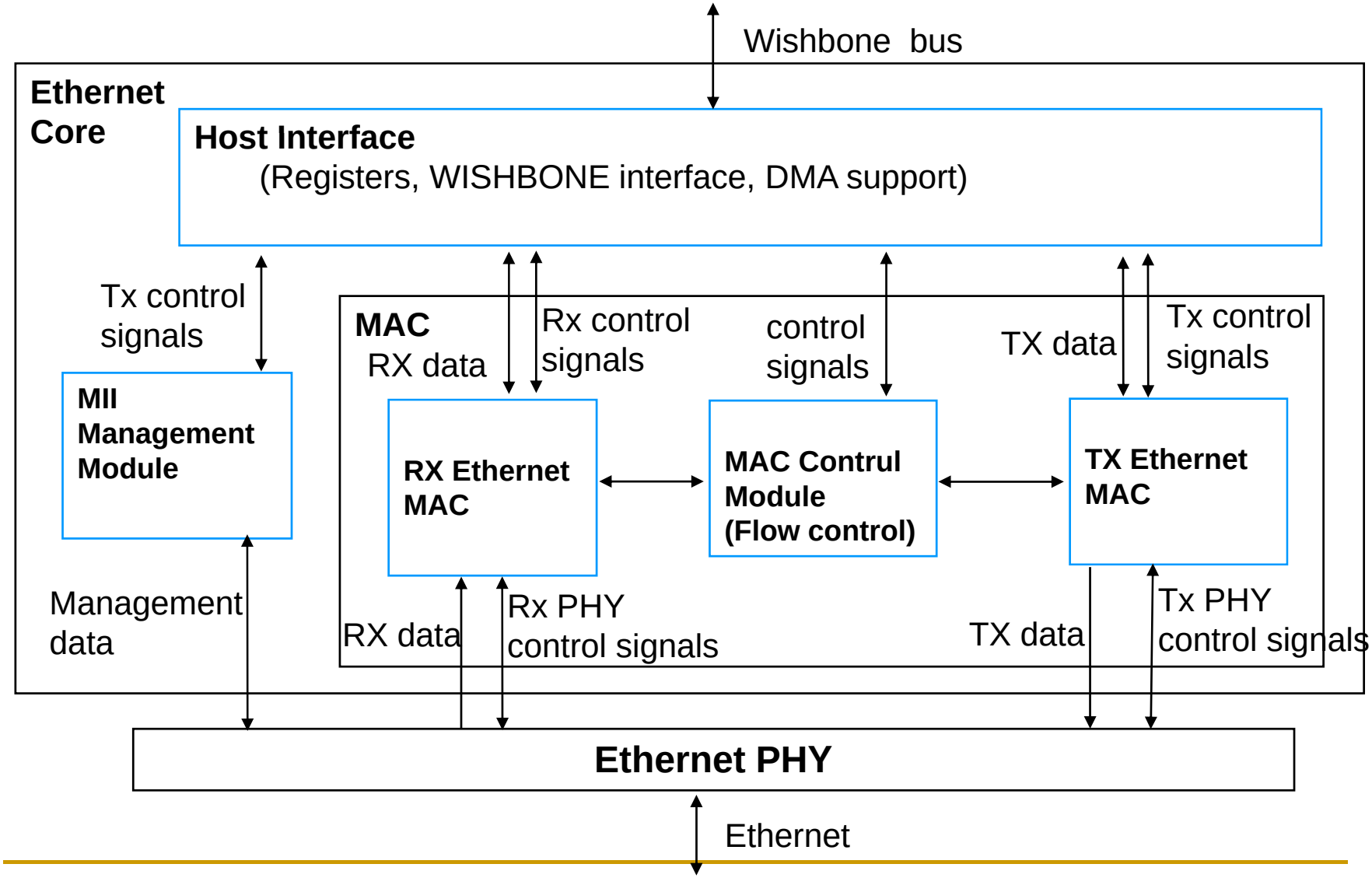
Open Source Implementation : Ethernet IP Core

- Totally five modules :
  - Host Interface Module
  - TX Ethernet MAC (transmit function)
  - RX Ethernet MAC (receive function)
  - MAC Control Module
  - MII Management Module

- Transmit, Receive, and MAC control modules form the MAC module

- For the complete Ethernet solution, an external PHY is needed

Architecture



Functions

- **Host Interface Module**
  - **Configuration registers**
  - **DMA operation**
  - **Transmit and receive status**

- **TX Ethernet MAC**
  - **Generation of control and status signals**
  - **Random time generation , used in the back-off process**
  - **Preamble generation**
  - **CRC generation**
  - **Pad generation**
  - **Data nibble generation**
  - **Inter Packet Gap**
  - **Monitoring CarrierSense and collision signals**

- **RX Ethernet MAC**
  - **Generation of control and status signals**
  - **Preamble removal**
  - **Data assembly**
  - **CRC checking**

Functions (cont.)

- **MAC Control Module**
  - **Control frame detection and generation**
  - **TX/RX MAC interface**
  - **PAUSE timer**
  - **Slot timer**

- **MII Management Module**
  - **Operation controller**
  - **Shift registers**
  - **Output control module**
  - **Clock generator**

I/O Ports

Host Interface ports (Signal direction is in respect to the Ethernet IP Core)

Port	Width	Directioin	Description
DATA_I	32	I	Data input
DATA_O	32	O	Data output
REQ0	1	O	DMA request to channel 0
REQ1	1	O	DMA request to channel 1
ACK0	1	I	DMA ack channel 0
ACK1	1	I	DMA ack channel 1
INTA_O	1	O	Interrupt output A

I/O Ports (cont.)

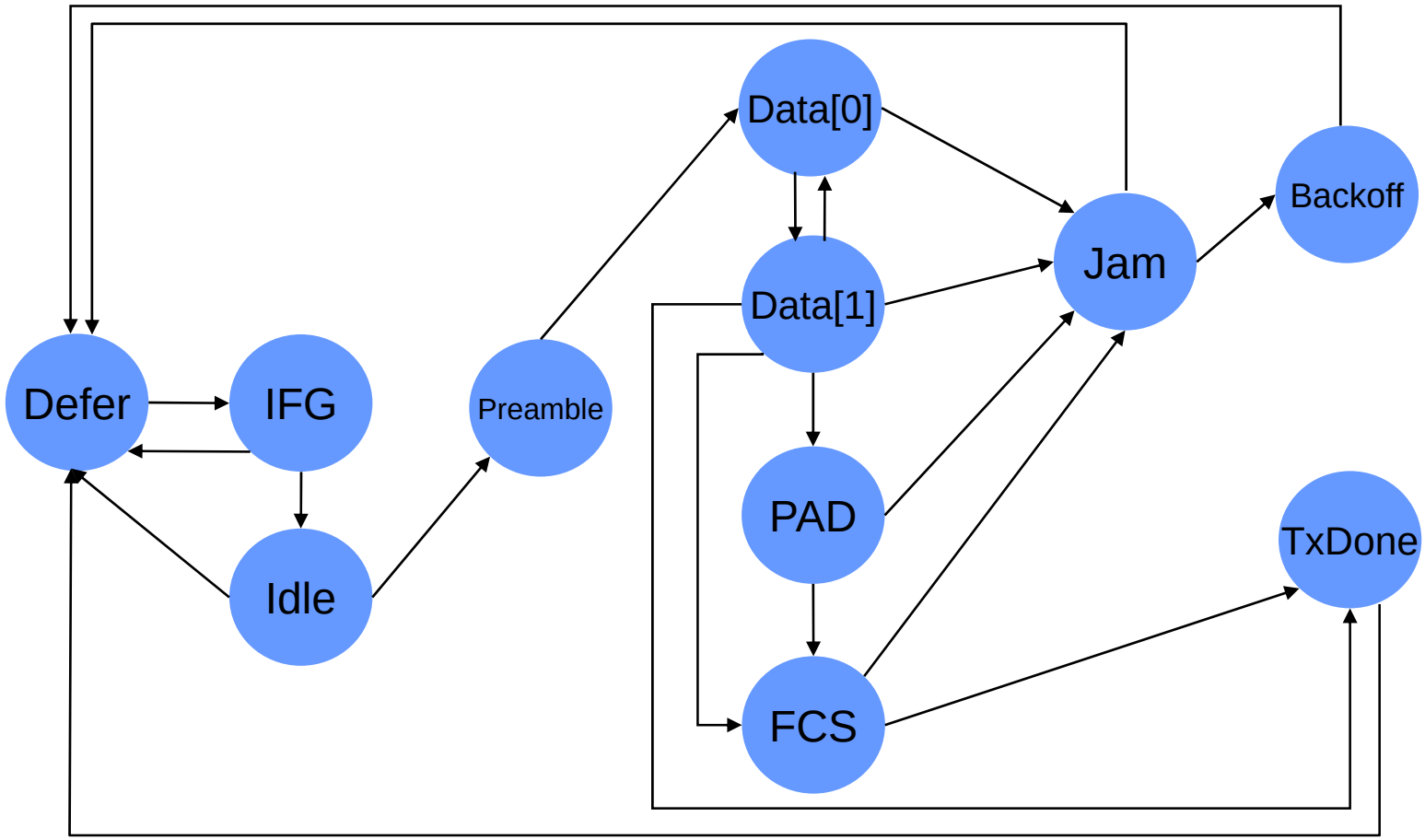
PHY Interface ports

Port	Width	Directioin	Description
MTxClK	1	I	Transmit nibble clock
MTxD[3:0]	4	O	Transmit data nibble
MTxEn	1	O	Transmit enable
MRxClK	1	I	Receive nibble clock
MRxDV	1	I	Receive data valid
MRxD[3:0]	4	I	Receive data nibble
MColl	1	I	Collision detected
MCrS	1	I	Carrier sense

Registers

Name	Address	Width	Access	Description
MODER	0x00	32	RW	Mode register
INT_SOURCE	0x01	32	RW	Interrupt source register
IPGT	0x03	32	RW	Inter packet gap register
PACKETLEN	0x06	32	RW	Packet length register
COLLCONF	0x07	32	RW	Collision and retry configuration
MAC_ADDR0	0x11	32	RW	MAC address (LSB 4 bytes)
MAC_ADDR1	0x12	32	RW	MAC address (MSB 2 bytes)

TX State Machine



CSMA/CD

- **CarrierSense** and **Collision** signals are provided from PHY

- assign StartDefer = StateIFG & ~Rule1 & **CarrierSense** & NibCnt[6:0] <= IPGR1 & NibCnt[6:0] != IPGR2
| StateIdle & **CarrierSense**
| StateJam & NibCntEq7 & (NoBckof | RandomEq0 | ~ColWindow | RetryMax)
| StateBackOff & (TxUnderRun | RandomEqByteCnt)
| StartTxDone | TooBig;
- assign StartData[1] = **~Collision** & StateData[0] & ~TxUnderRun & ~MaxFrame;

- assign StartJam = (**Collision** | UnderRun) & ((StatePreamble & NibCntEq15) |(|StateData[1:0]) | StatePAD | StateFCS);

- assign StartBackoff = StateJam & ~RandomEq0 & ColWindow & ~RetryMax & NibCntEq7 & ~NoBckof;

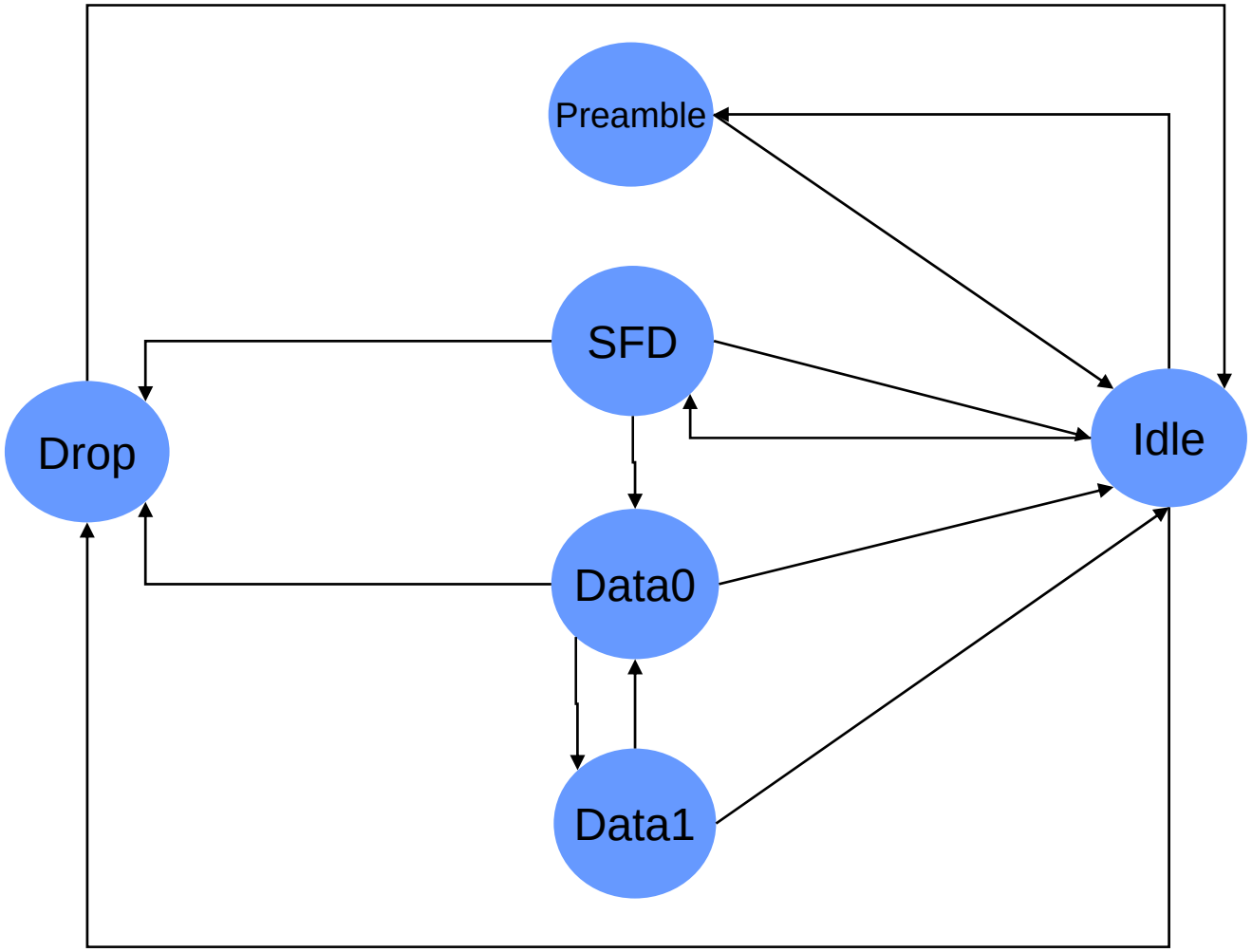
Transmit Nibble

```
always @ (StatePreamble or StateData or StateData or StateFCS or StateJam or
                StateSFD or TxData or  Crc or NibCnt or NibCntEq15)
begin
   if(StateData[0])    MTxD_d[3:0] = TxData[3:0];          // Lower nibble
   else
   if(StateData[1])    MTxD_d[3:0] = TxData[7:4];          // Higher nibble
   else
   if(StateFCS)    MTxD_d[3:0] = {~Crc[28], ~Crc[29], ~Crc[30], ~Crc[31]};     // Crc
   else
   if(StateJam)    MTxD_d[3:0] = 4'h9;                     // Jam pattern
   else
   if(StatePreamble)
       if(NibCntEq15)      MTxD_d[3:0] = 4'hd;             // SFD
       else      MTxD_d[3:0] = 4'h5;                       // Preamble
   else    MTxD_d[3:0] = 4'h0;
end
```

Back-off Random Generator

```
assign Random [0] = x[0];
assign Random [1] = (RetryCnt > 1) ? x[1] : 1'b0;
assign Random [2] = (RetryCnt > 2) ? x[2] : 1'b0;
assign Random [3] = (RetryCnt > 3) ? x[3] : 1'b0;
assign Random [4] = (RetryCnt > 4) ? x[4] : 1'b0;
assign Random [5] = (RetryCnt > 5) ? x[5] : 1'b0;
assign Random [6] = (RetryCnt > 6) ? x[6] : 1'b0;
assign Random [7] = (RetryCnt > 7) ? x[7] : 1'b0;
assign Random [8] = (RetryCnt > 8) ? x[8] : 1'b0;
assign Random [9] = (RetryCnt > 9) ? x[9] : 1'b0;
always @ (posedge MTxClk or posedge Reset)
begin
  if(Reset)
    RandomLatched <= 10'h000;
  else
    begin
      if(StateJam & StateJam_q)
        RandomLatched <= Random;
    end
end
assign RandomEq0 = RandomLatched == 10'h0;
```

RX State Machine



RX Error Checking

// CRC and reception error checking
- assign ReceivedPacketGood = ~LatchedCrcError & ~LatchedMRxErr;

// frame size checking
- assign ReceivedLengthOK = LatchedRxByteCnt[15:0] > 63 &
  LatchedRxByteCnt[15:0] < 1519;

- Address Detection and checking
  - MAC address is from register