

# Cognitive Wireless Networking with WARP

## Part – II: Reconfigurable MAC Design

Junaid Ansari and Xi Zhang

Institute for Networked Systems  
RWTH Aachen University, Germany

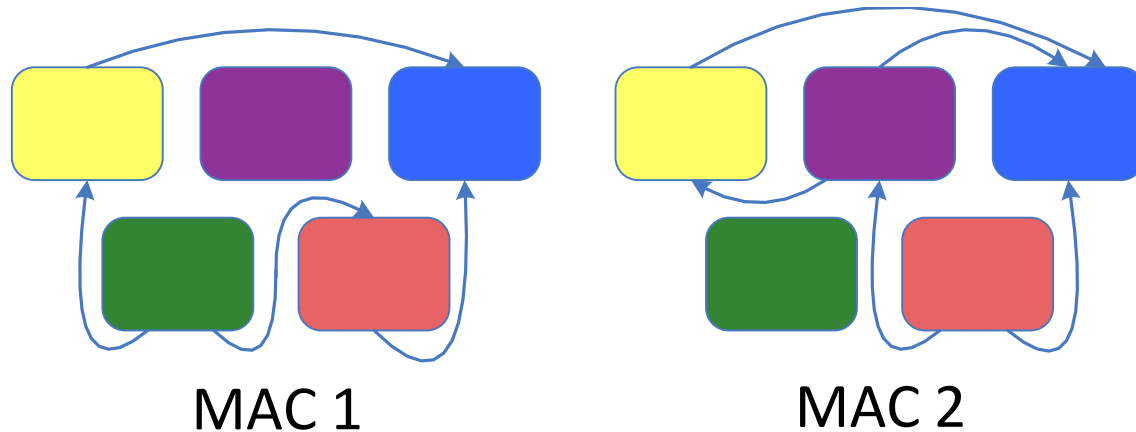
3<sup>rd</sup> May 2011

# Motivation

- Cognitive MAC protocols require **fine-grained** access control over the PHY/MAC parameters and **run-time reconfiguration**.
- **Hardware**-based MAC implementations are **rigid** and do not provide the required flexibility.
- **Software**-based implementations on the contrary **fail to meet strict timing deadlines**.
- A **hardware-software co-design** approach is desired in order to simultaneously meet the timing deadlines and provide the required flexibility.

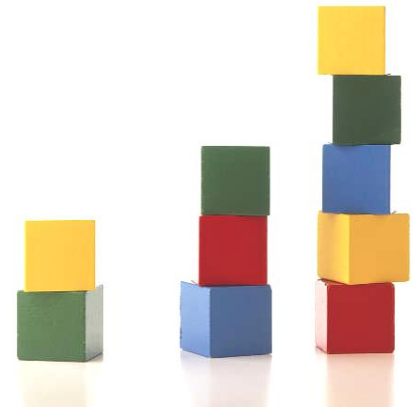
# Component Oriented Design

- Concept: MAC protocols are **decomposed** into fundamental functional components based on the **commonalities** among different protocols.
- These components serve as the **MAC building blocks**.
- A particular MAC protocol is realized by (simply) **binding** these **components** together.



# Fundamental MAC Building Blocks

- Timer functionalities
- Carrier sensing algorithms
- Radio state control
- Random number generation
- Framing and buffer management
- Sending frame
- Receiving frame
- ...

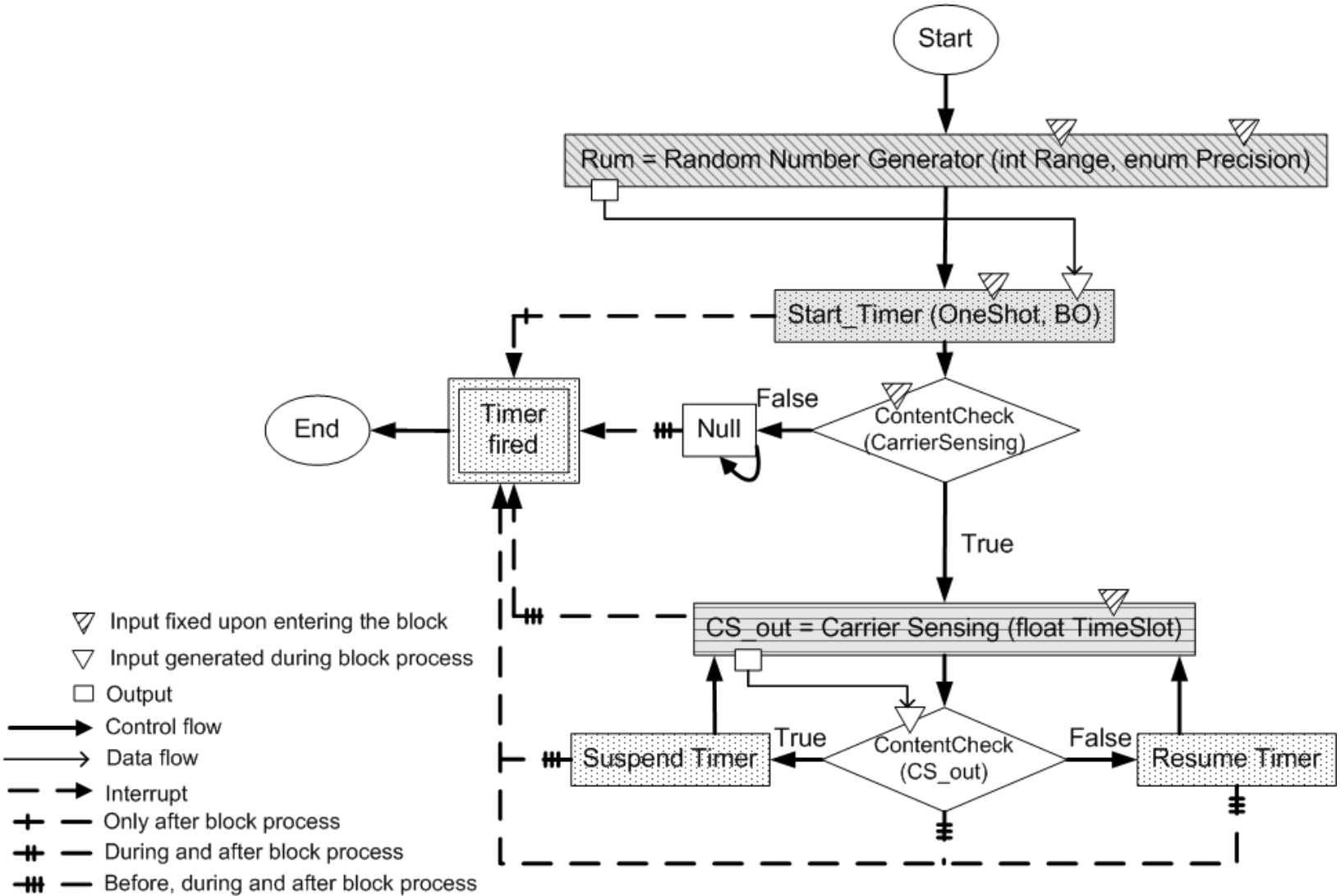


# Advanced Building Blocks

- Certain **combinations** and **patterns** of fundamental blocks **repeat** across different protocol implementations.
- This leads to the concept of “**Blocks of Blocks**” or **secondary blocks**.
- The design philosophy is similar to **LEGOS**.



# Random Backoff – a Closer Look

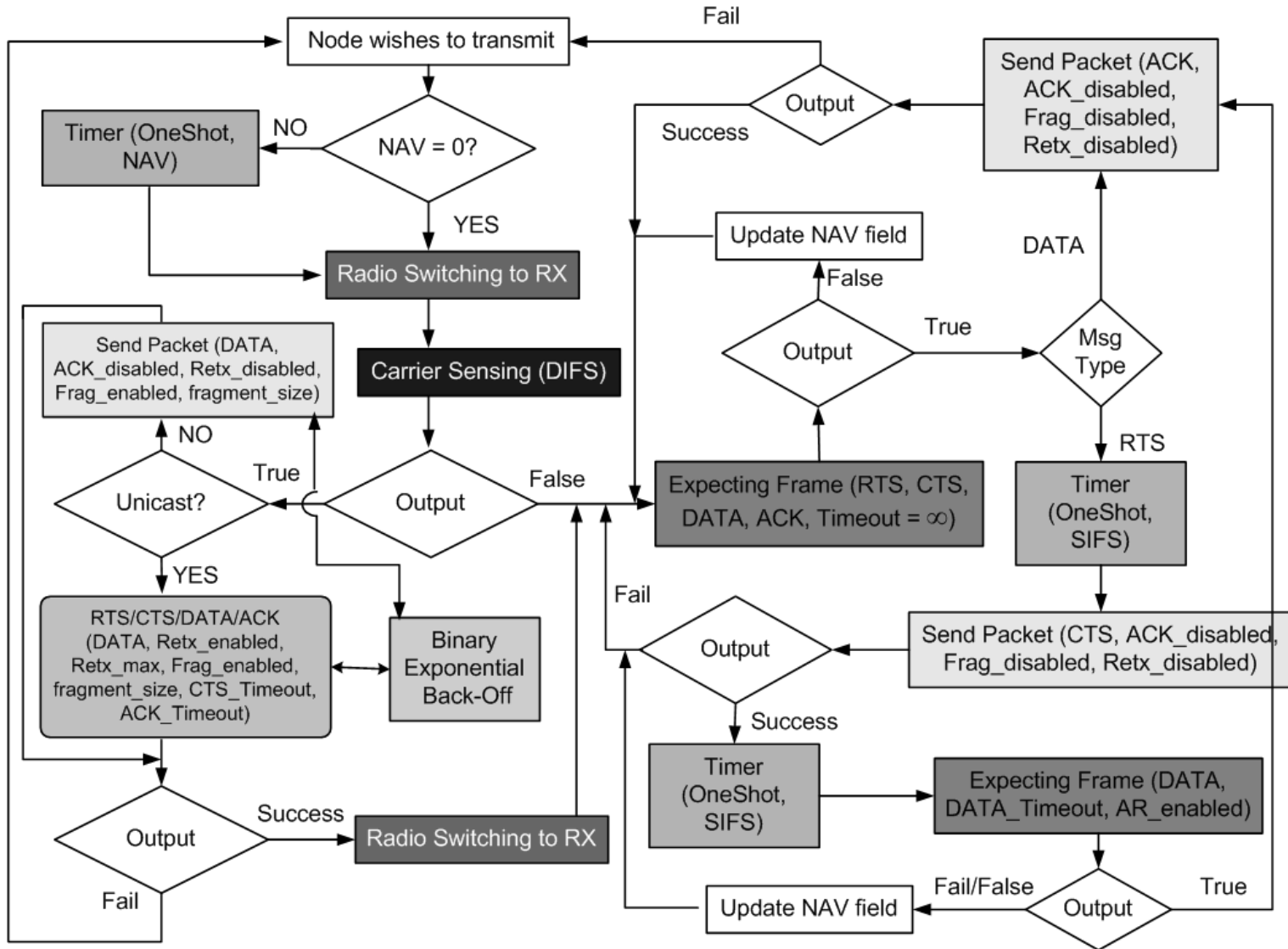


# Commonly Used Secondary Blocks

## COMMONLY USED SECONDARY LEVEL MAC COMPONENTS.

Component	Usage and the <i>composition</i>
<i>Random Backoff</i>	Random backoff mechanism <i>Timer, Random Number Generator, Carrier Sensing</i>
<i>Expecting Frame</i>	Used when the node is waiting in anticipation of a packet <i>ReceiveFrame, Timer, Radio Switching, SendFrame</i>
<i>Send Packet</i>	Called after seizing a channel free <i>SendFrame, Expecting Frame, Radio Switching, Random backoff</i>
<i>RTS/CTS/DATA/ACK</i>	Four-way handshake mechanism <i>Send Packet, Expecting Frame</i>

# Realization of IEEE 802.11 DCF



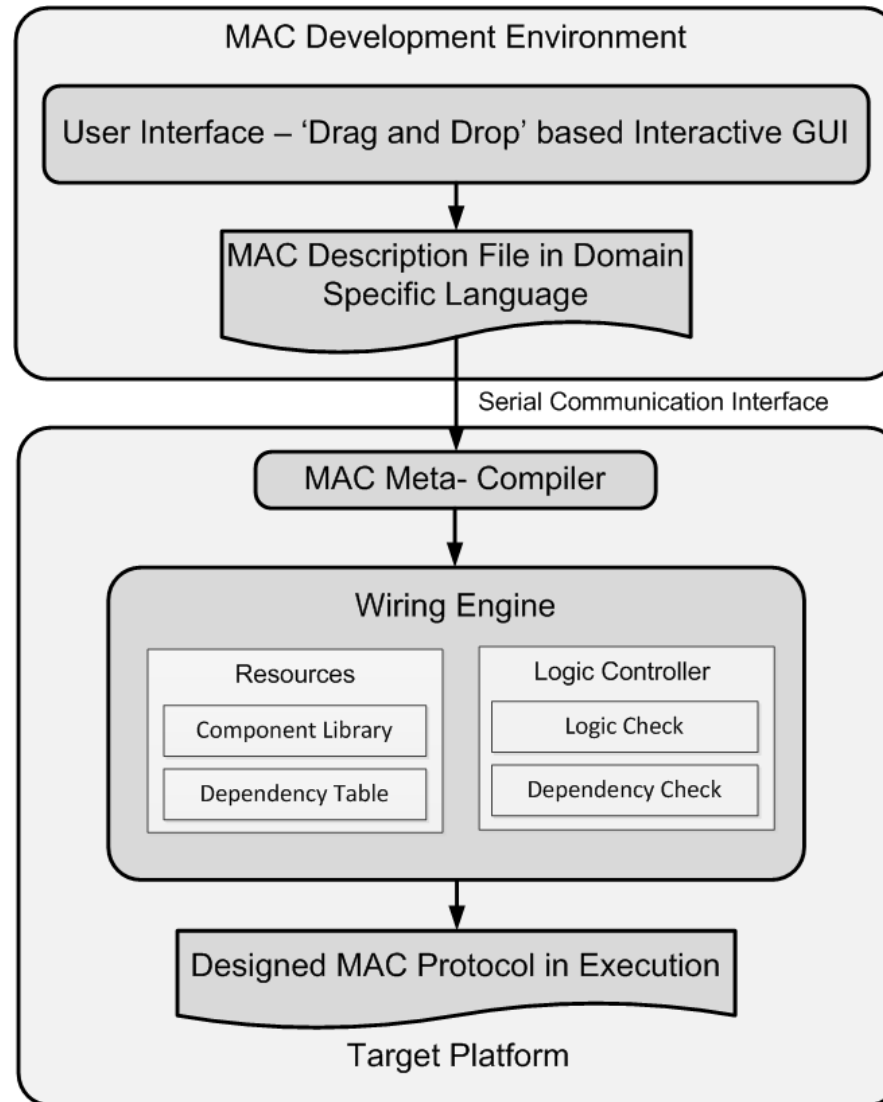


# MAC Realization Toolchain

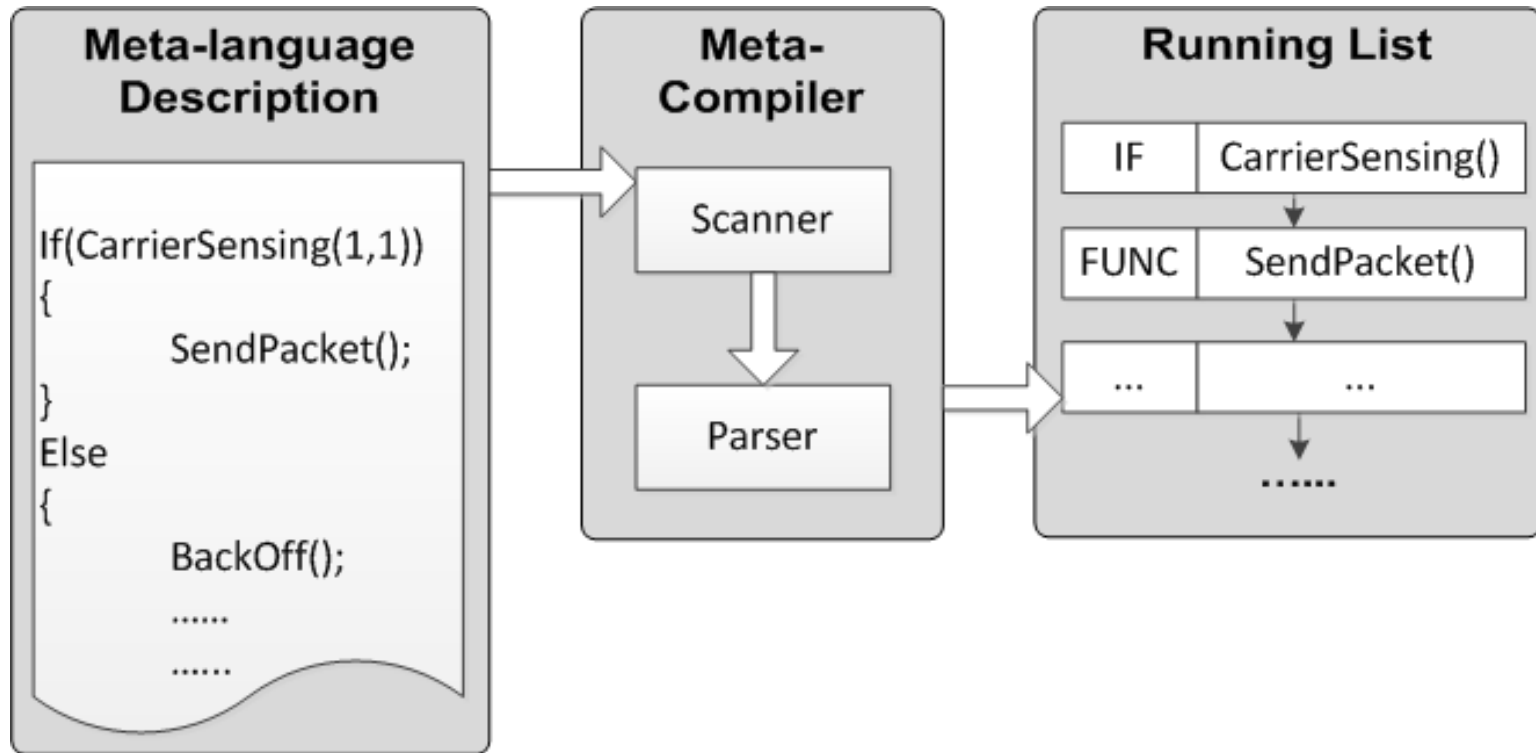
- **Wiring Engine** to bind the MAC components and coordinate the control and data flow among components.
- **MAC meta-language** to describe the MAC design.
- A (host) **compiler** to convert the MAC language to executable code on a particular target platform.
- Interactive **Graphical User Interface (GUI)** to ease the MAC designing process.



# Toolchain Assisted MAC Designing Process



# MAC Compilation and Execution



# MAC Meta-language

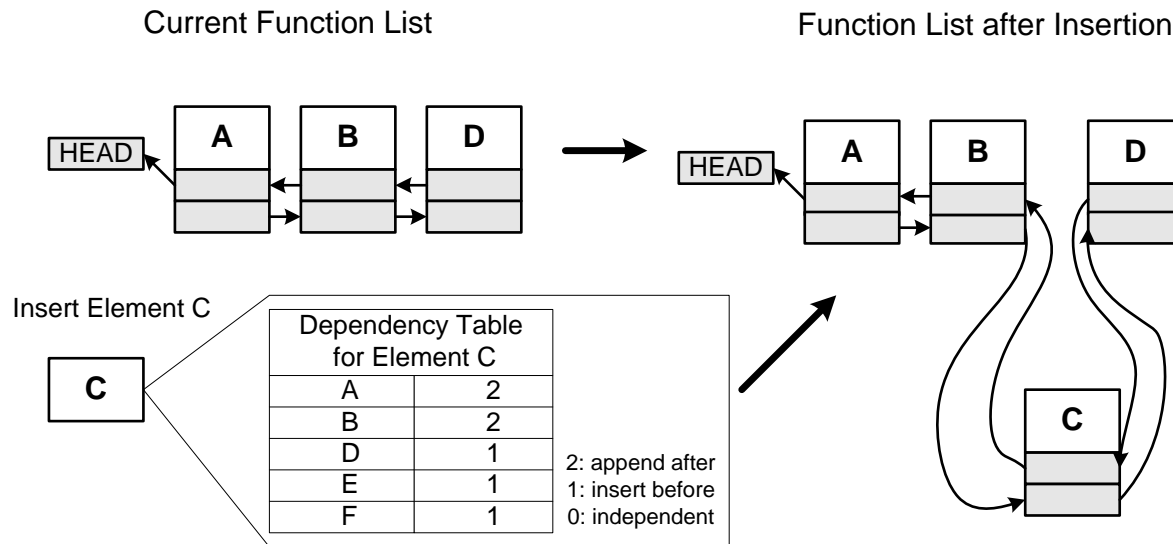
- C-like syntax
  - Variable and constant declarations: `VAR`, `CONST`.
  - Conditional branching: `IF`, `ELSE`, `ENDIF`.
  - Loops: `LABEL`, `GOTO`.
- All the MAC functions are wrapped with a standard component API: `int function (void *para)`.
- Extendable grammar and functions.

# Meta-compiler

- A **scanner** to scan the program file to recognize keywords and tokens.
- A **parser** to determine the grammatical structure and checks for syntax errors.
- A **code generator** for generating executable code accordingly for the target platform.
- Compiler is written using Lex & Yacc.

# Rapid Protocol Reconfiguration

- Modifying protocol == modifying function linked list.
- Allows on-the-fly re-configuration by block **insertion**, **removal** and **re-wiring** w.r.t. their **dependency tables**.



- Built-in **optimizer** assisted reconfiguration.
- **User triggered** reconfiguration.

# IDE for Rapid Protocol Development

The screenshot displays the IDE interface for 'Trump GUI for WARP Board MAC Protocol Design'. The main workspace is the **Designer Pane**, which contains a flowchart. The flowchart starts with a yellow oval labeled 'Label:1', followed by a purple rectangular block 'Function: Timer( Create,Timer1,100,ONE\_SHOT,Millisecond)'. Below this is a green diamond-shaped decision block 'IF: CarrierSensing(10,10)'. The 'TRUE' path leads to a purple block 'Function: NewPacket(DATA,1,100)', which then leads to another purple block 'Function: SendPacket()'. The 'FALSE' path leads to a yellow rectangular block 'Goto: Label 1', which loops back to the 'Label:1' oval. The flowchart ends with a green rectangular block 'End-IF'.

On the right side, the **TOOLS PANE** contains various symbols for building the flowchart: Select (blue dots), Function Block (purple rectangle), Label (yellow oval), Goto (yellow rectangle), Goto Edge (purple arrow), Follow (grey arrow), IF-ELSE (green diamond), END-IF (green rectangle), and Expression (grey rectangle).

At the bottom left, the **Source Code** pane shows the following code:

```
0: LABEL    label 1;
1: FUNC    Timer(Create, Timer1, 100, ONE_SH
2: IF      if( CarrierSensing(10, 10) ) {
3: FUNC    NewPacket(DATA, 1, 100);
4: FUNC    SendPacket();
5: ELSE
6: GOTO    goto 1;
7: END     endif
```

At the bottom right, the **Console** pane shows the following output:

```
Console: Connected to COM7
tx rx throughput 2320 0 at channel 5
tx rx throughput 2240 0 at channel 5

Rssi = 114 tx rx throughput 2240 0 at channel 5
tx rx throughput 2240 0 at channel 5
tx rx throughput 2240 0 at channel 5
File transmission finished!
received command STOP
Stopping succeeded.
```

Below the console are buttons for 'Disconnect', 'New', 'New File', 'Clean', 'Print', 'Run', and 'Stop', along with a 'Command:' input field and a 'Send' button.

Flow control and  
MAC design logic

Designer Pane

Generated  
code in MAC  
language

Communication  
with target platform

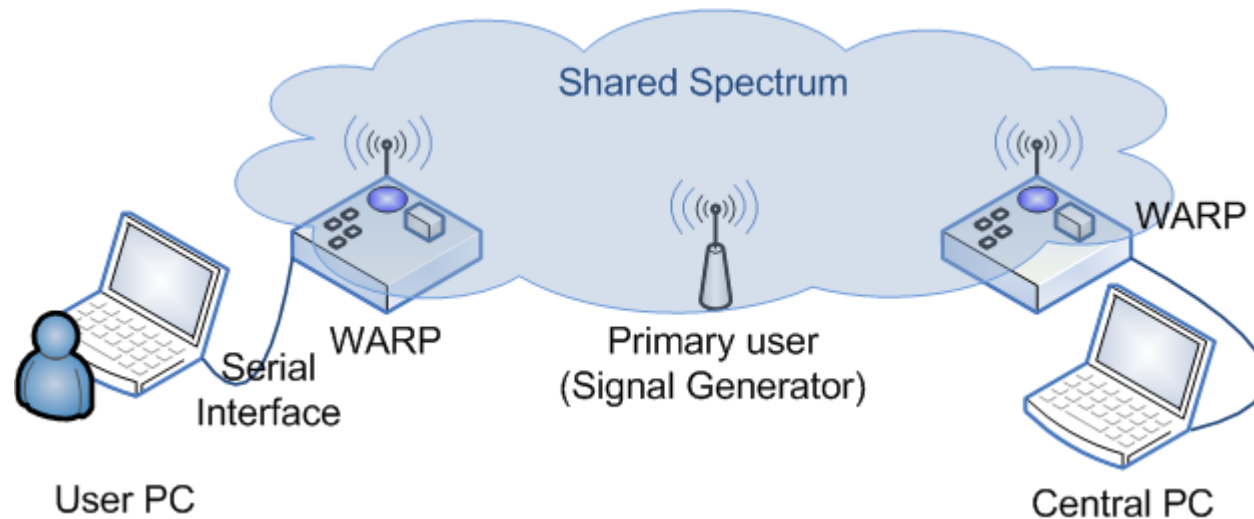
# In a nut-shell...

- Enables **fast** protocol development.
- Allows **code re-use** and minimizes efforts.
- Opens **wider experimental room**.
- Enables **run-time reconfiguration**.



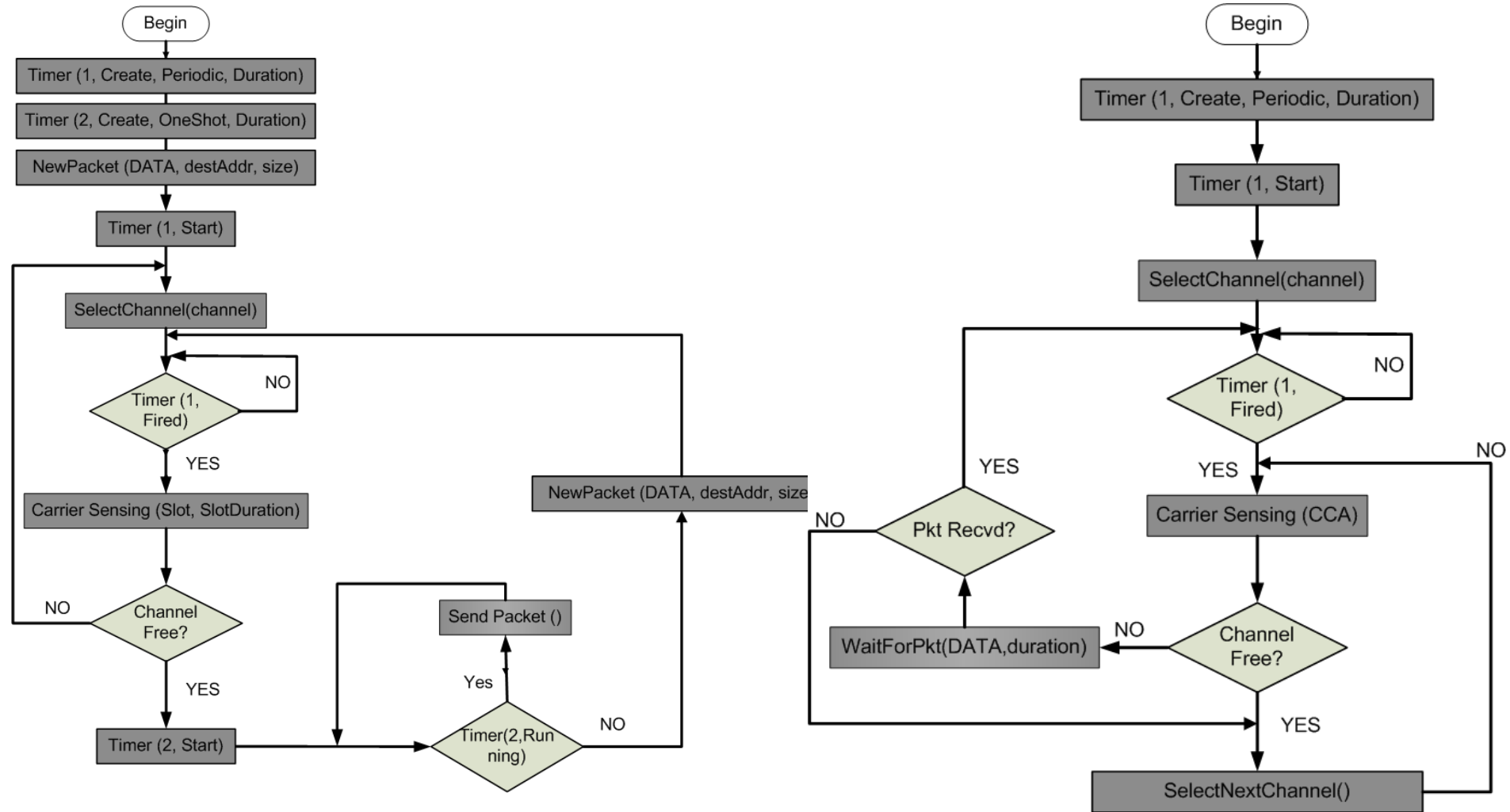
# Let's get our hands dirty...

- You will get the handouts for detailed description ...
- Task: Developing a simple Spectrum Agile MAC



WARP Laboratory Setup

# Transmitter/Receiver Design



# Component APIs

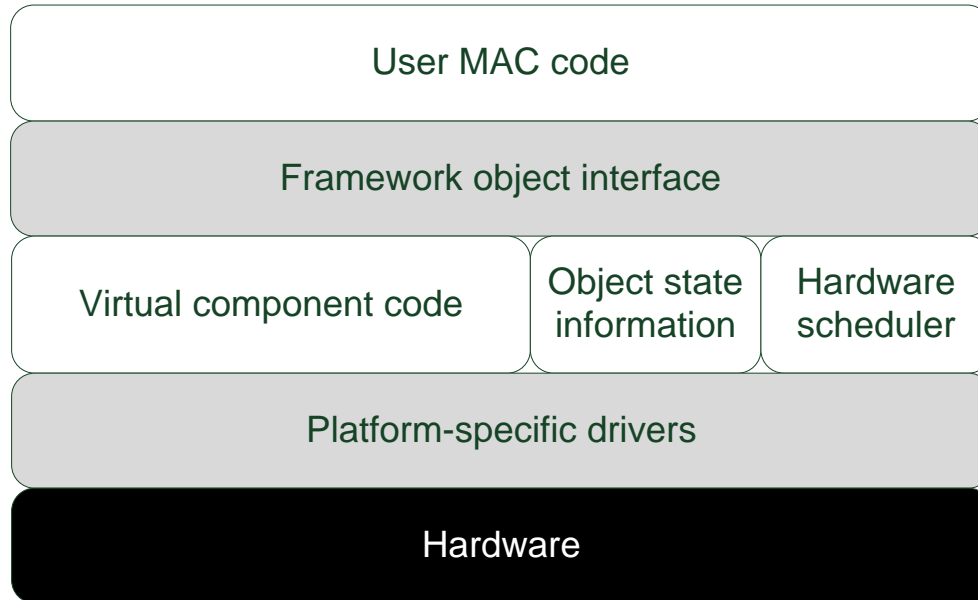
Packet Creation
<b>Functional API:</b> <code>int NewPacket(int pktType, int pktDest, int pktSize);</code>
<b>Inputs:</b> <ul style="list-style-type: none"><li>• Packet type: <code>int pktType</code> (DATA = 0; ACK = 1)</li><li>• Packet destination: <code>int pktDest</code></li><li>• Packet size: <code>int pktSize</code> in Bytes</li></ul>
<b>Return value:</b> SUCCESS = 1; FAIL = 0
<b>Description:</b> Create a packet to the destination address with assigned packet size and packet type with a unique sequence number.

Packet Transmission
<b>Functional API:</b> <code>int SendPacket();</code>
<b>Inputs:</b> None
<b>Return value:</b> SUCCESS = 1; FAIL = 0
<b>Description:</b> Send a packet which has been previously created by function <code>NewPacket</code> .

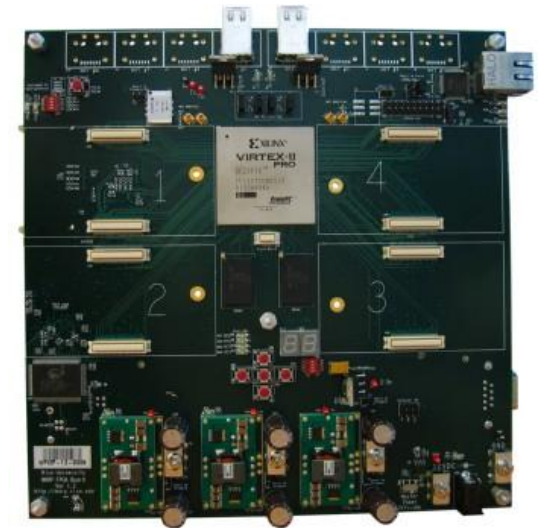
Questions?

Backup slides

# Implementation Details on WARP

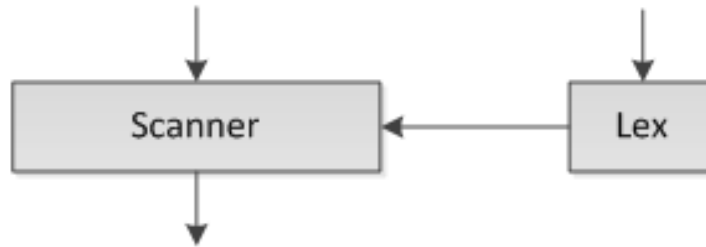


© <http://warp.rice.edu/trac>

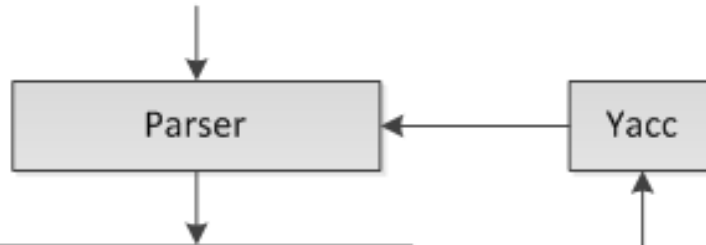


# Meta-compiler -cntd.-

Source code: `if( CarrierSensing( 1, 1 )`      Token Definitions



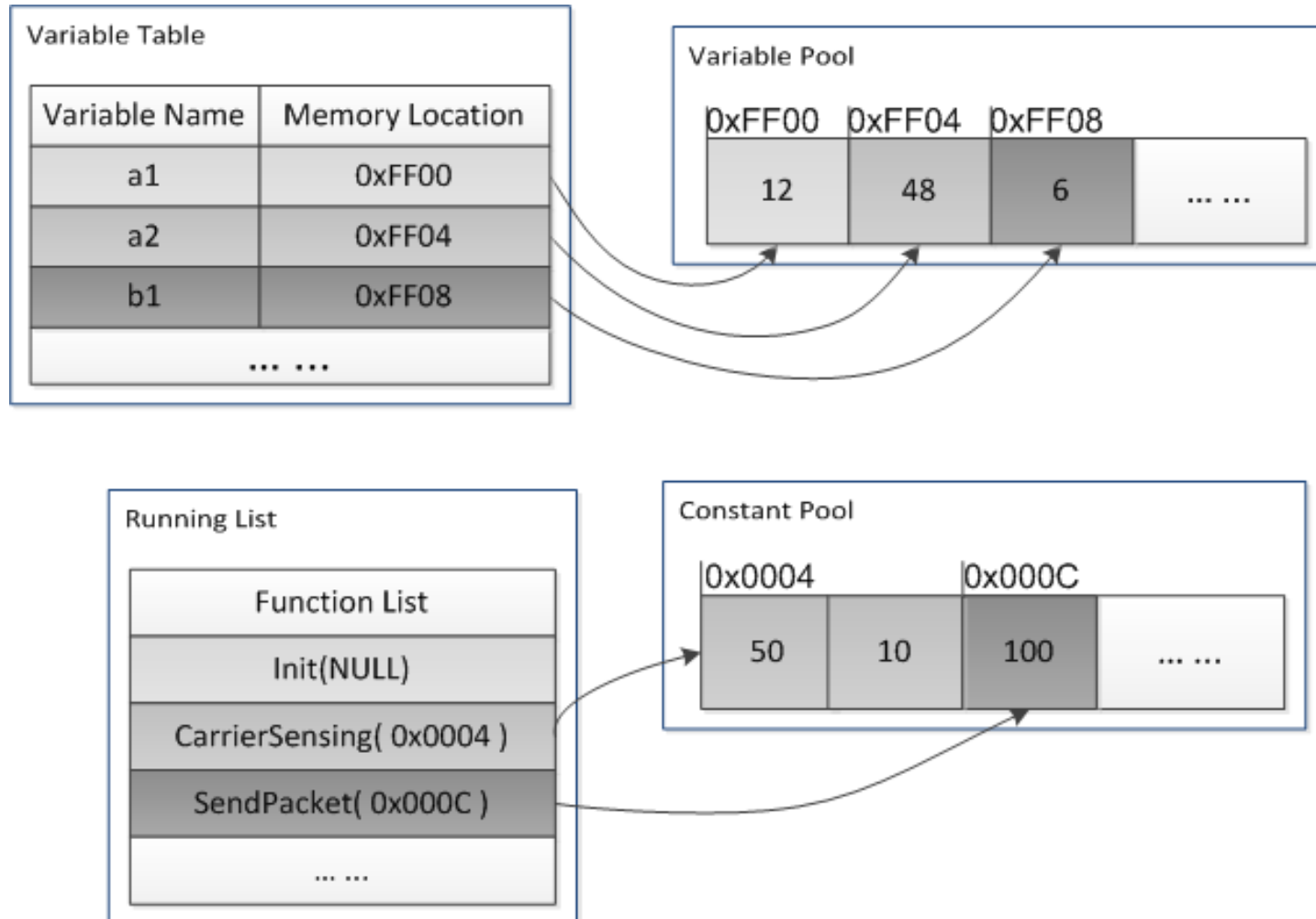
Tokens: `if '(' function '(' constant ',' constant ')' '('`



Node Generated
Type: IF
Function pointer: CarrierSensing
Parameter: 1, 1
Return value: 0
Table Index: 2

Grammar Definitions

# Memory Management





# Example: Timer Interface

- Commands:
  - `start()` starts a timer.
  - `stop()` stops a timer if the timer is not expired yet.
  - `suspend()` suspends the running of a timer until it is resumed.
  - `resume()` resumes the running of a timer after the suspension.
  - `getStart()` returns start time of the timer
  - `getDuration()` returns timer duration
  - `getStatus()` returns the current status of a timer (running, suspended, etc.)
- Input Parameters:
  - Type
    - One shot timer
    - Periodic timer
  - Precision
    - Millisecond
    - Microsecond
- Output:
  - Signals when timer expires.