Cognitive Wireless Networking with WARP

Part – II: Reconfigurable MAC Design

Junaid Ansari and Xi Zhang

Institute for Networked Systems

RWTH Aachen University, Germany

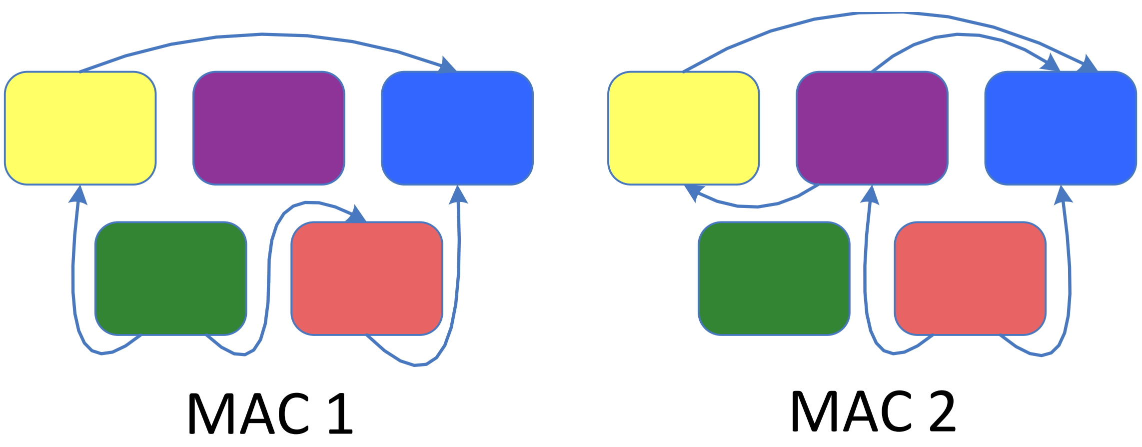
3rd May 2011

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.

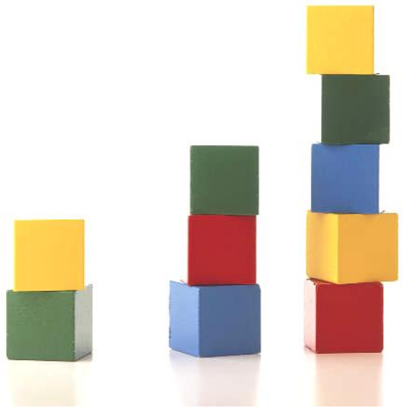
Component Oriented Design

- 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

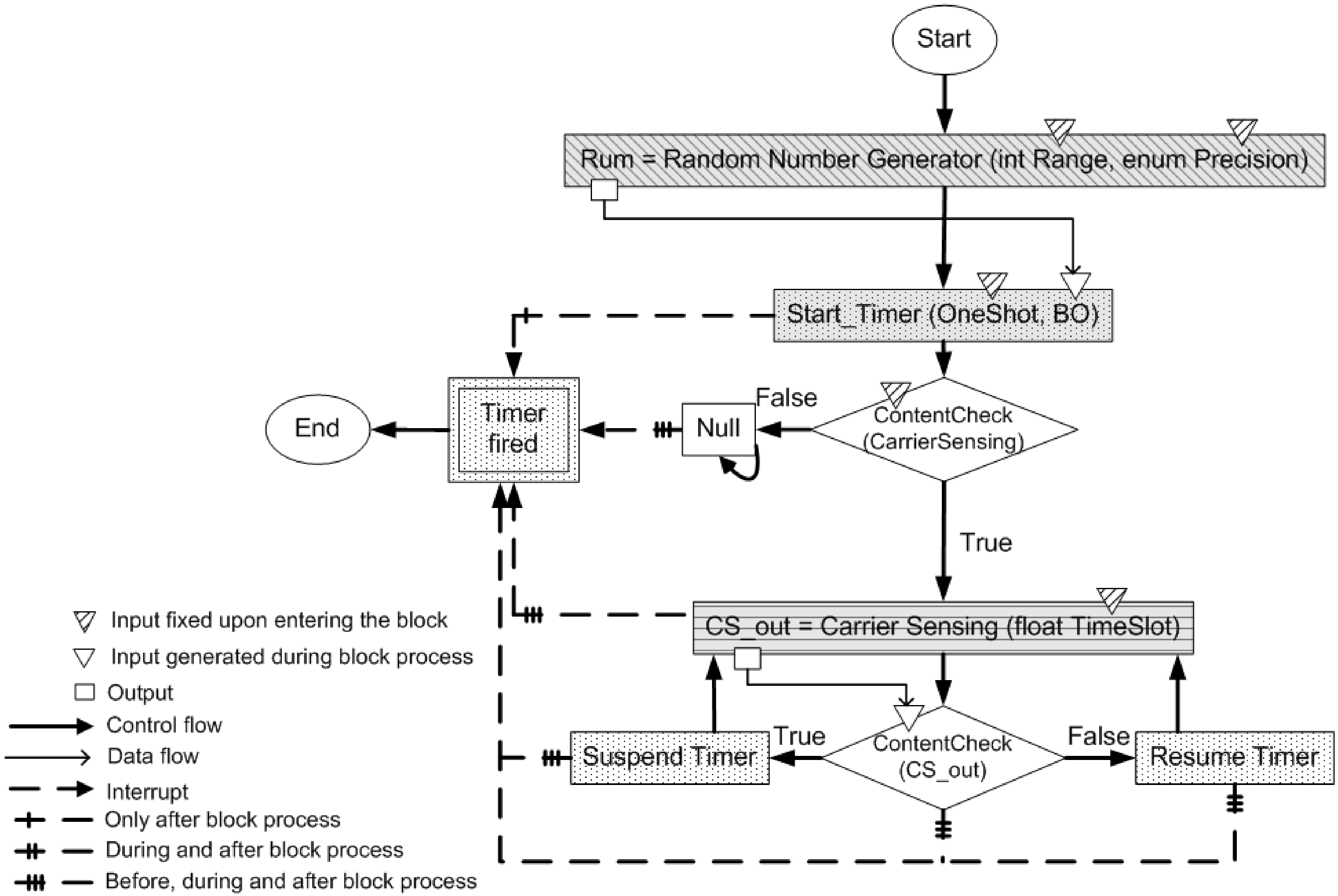
- Timer functionalities
- Carrier sensing algorithms
- Radio state control
- Random number generation
- Framing and buffer management
- Sending frame
- Receiving frame
- …



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

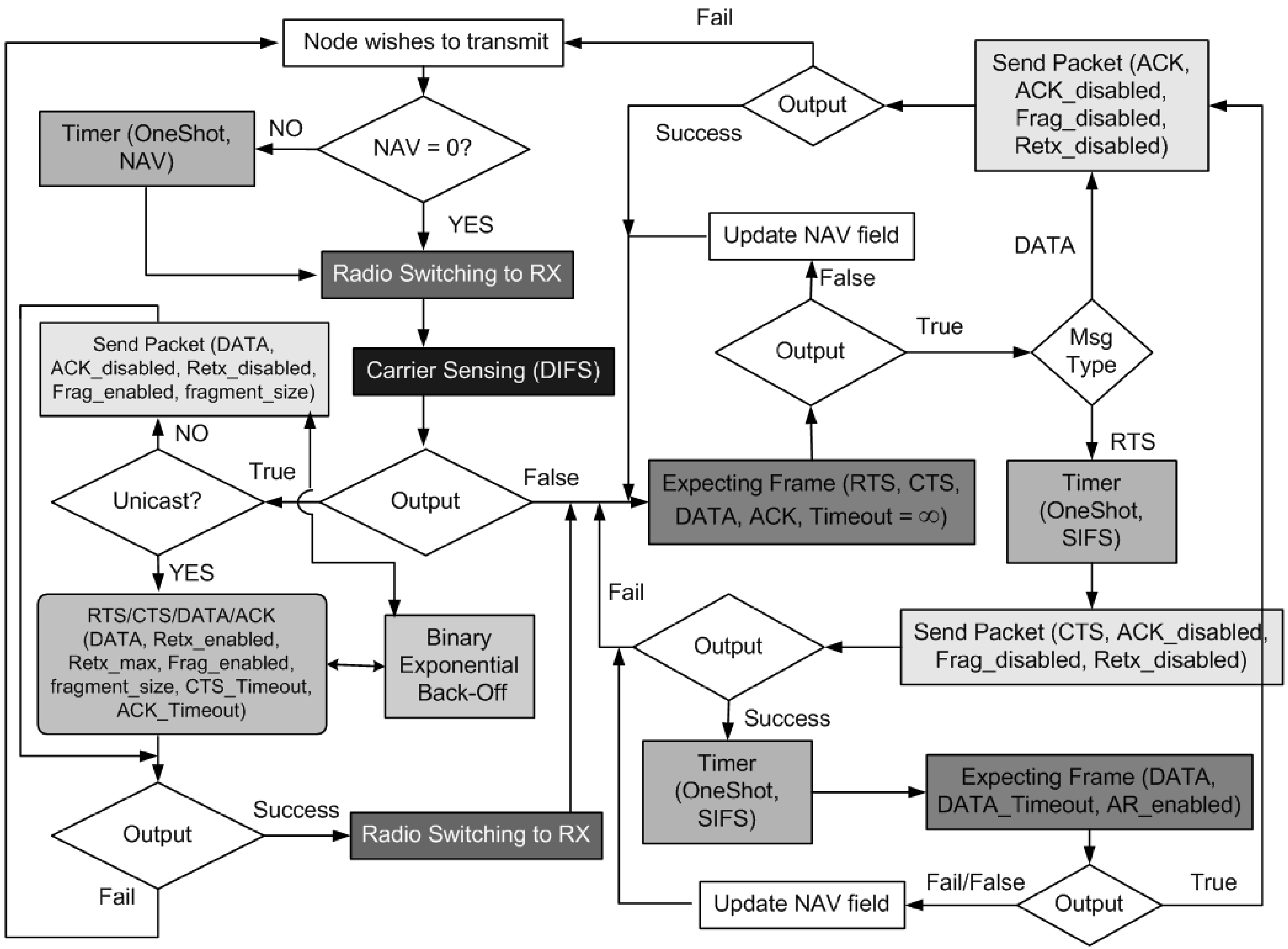


Commonly Used Secondary Blocks

COMMONLY USED SECONDARY LEVEL MAC COMPONENTS.

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

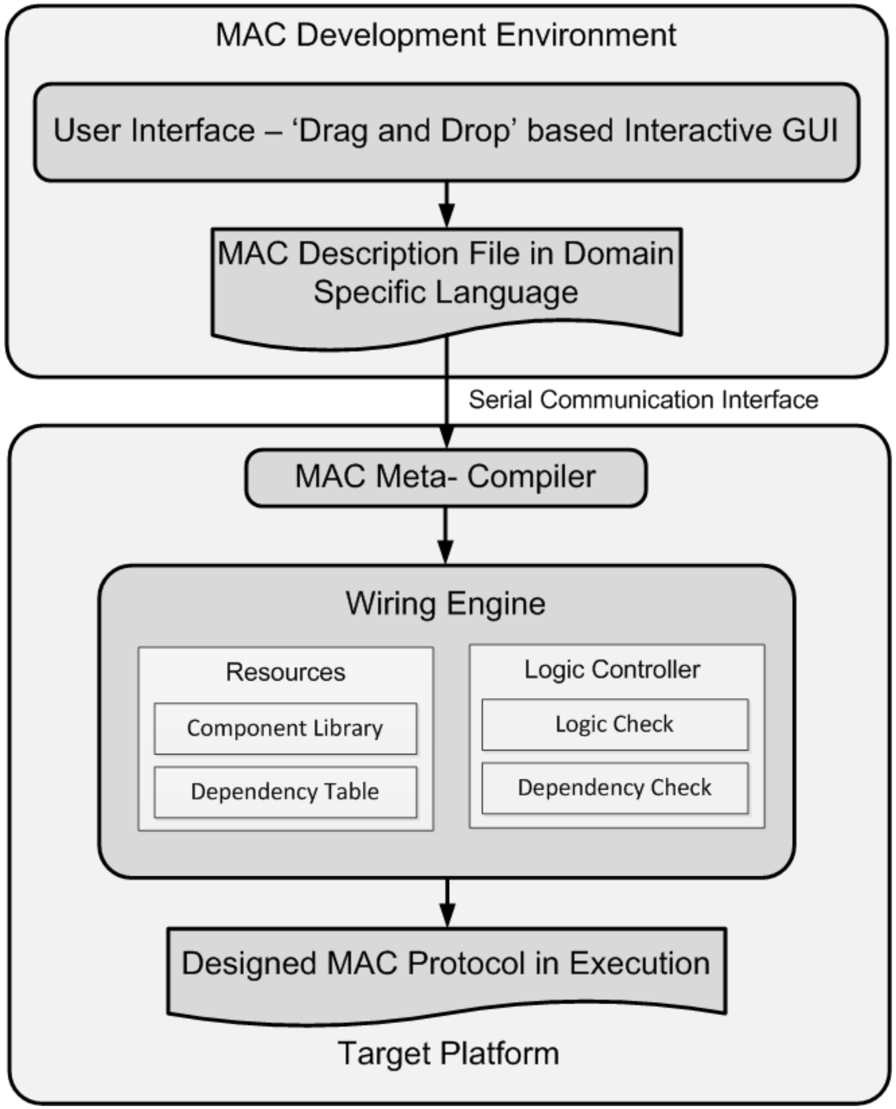
Realization of IEEE 802.11 DCF



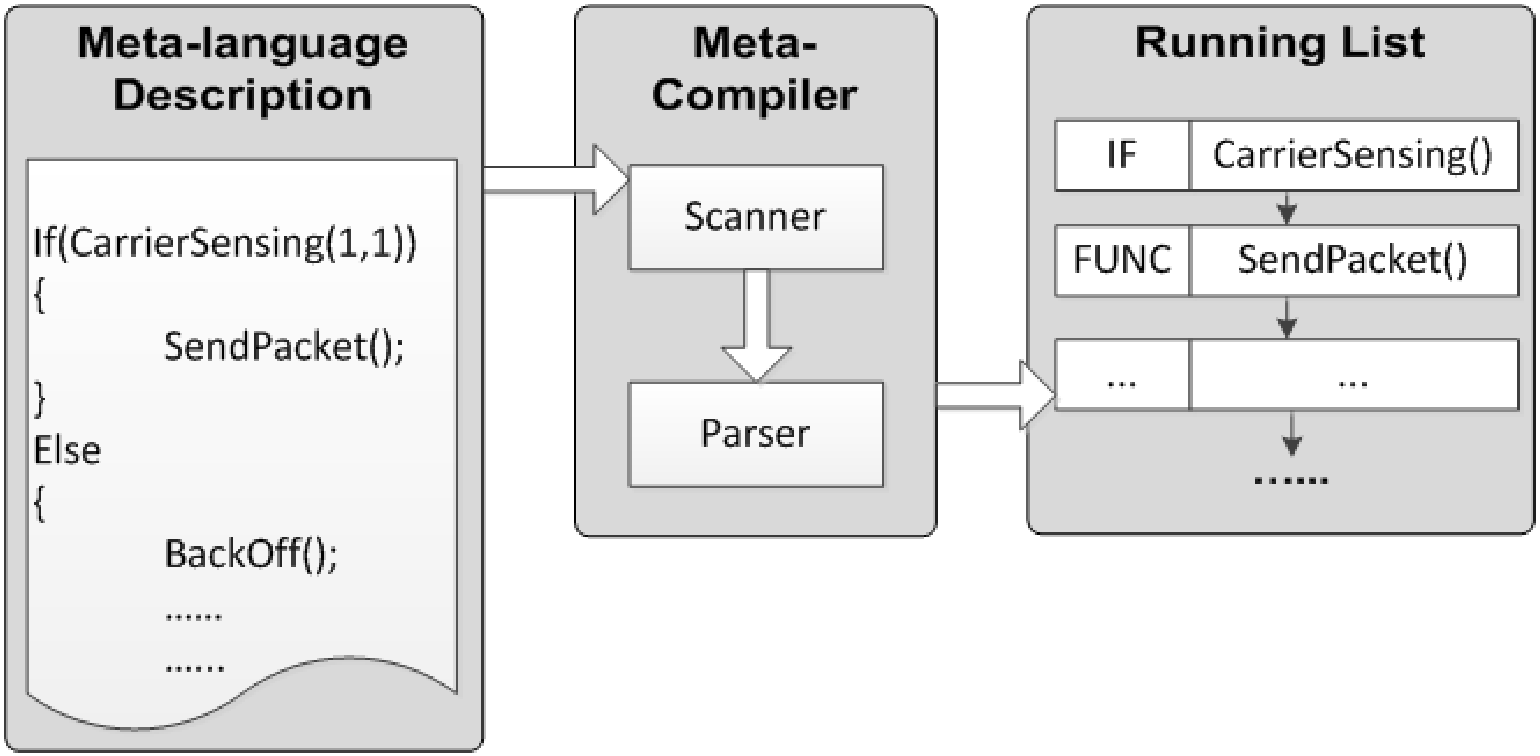
MAC Realization Toolchain

- 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).

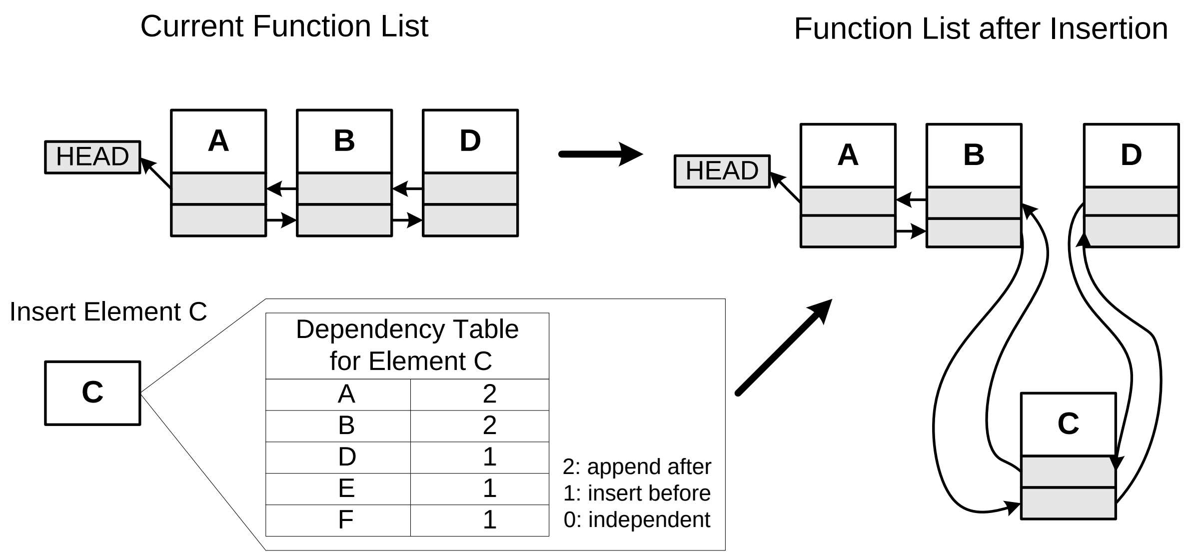
- Extendable grammar and functions.

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.

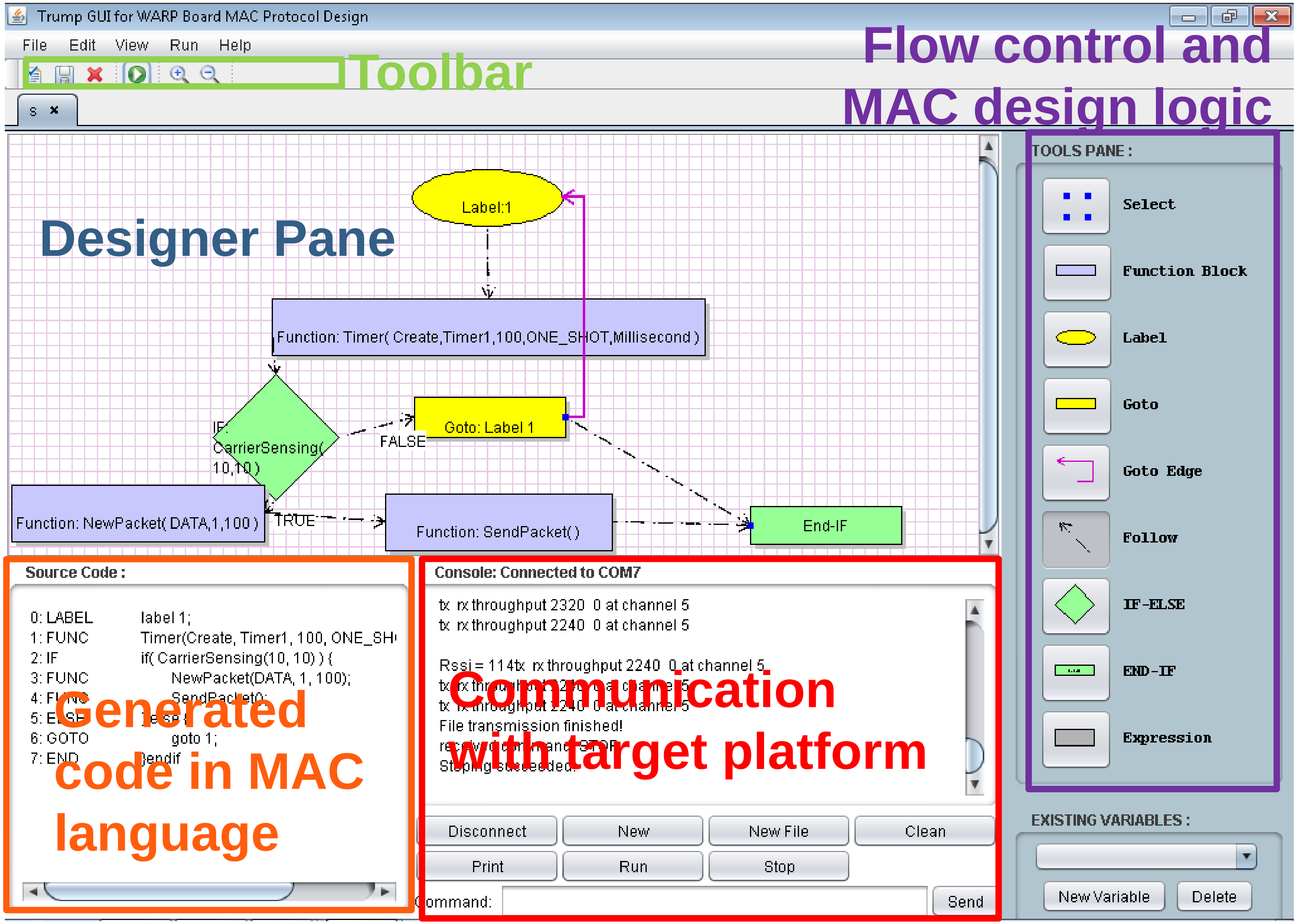
Rapid Protocol Reconfiguration

- 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

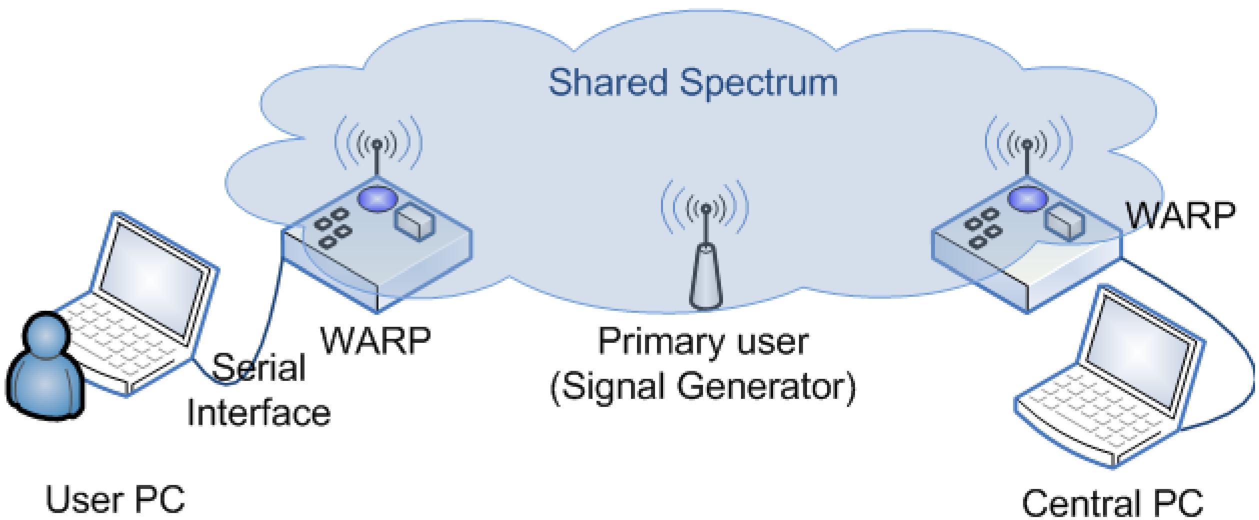


In a nut-shell…

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

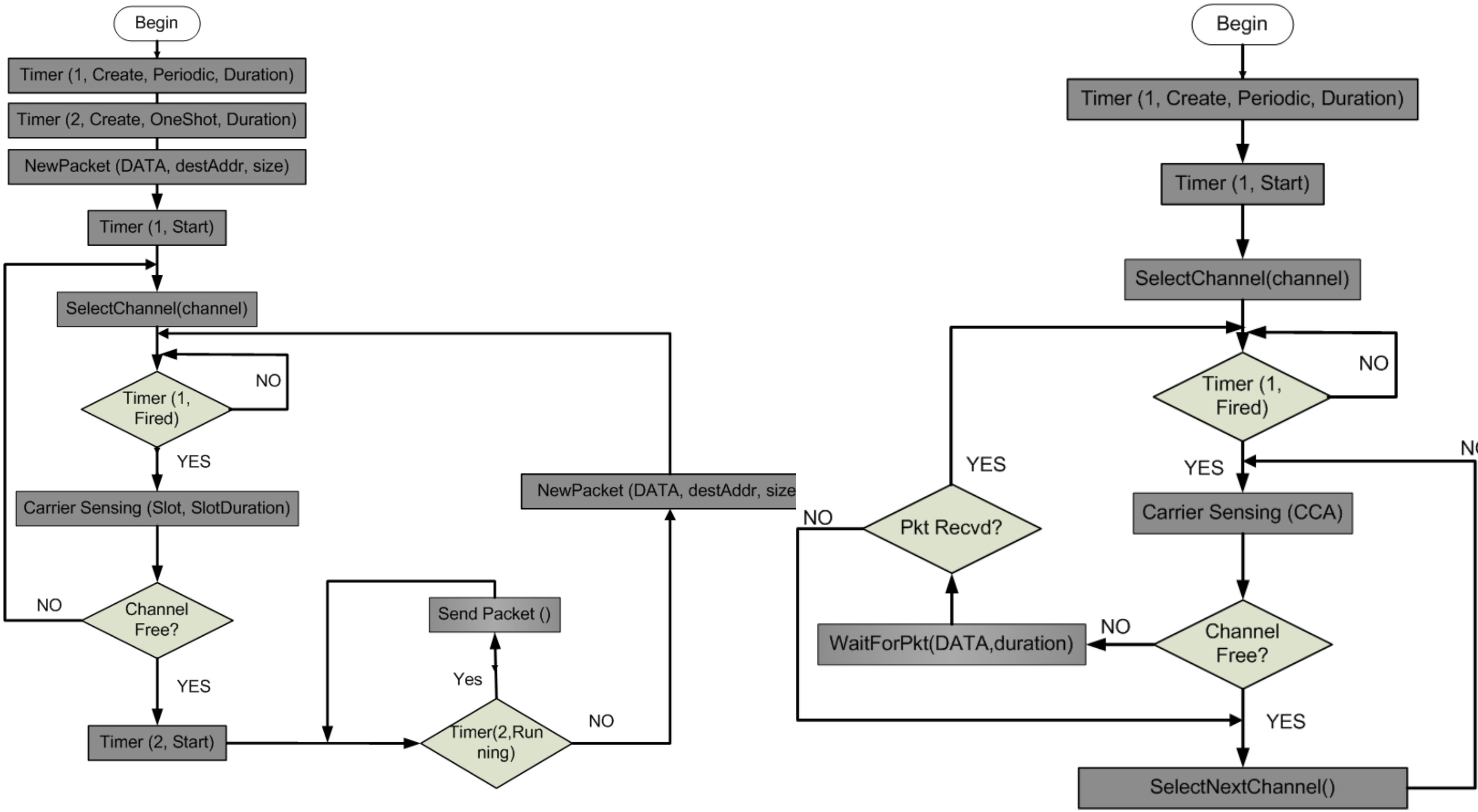
Let's get our hands dirty…

- 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
Functional API: `int NewPacket(int pktType, int pktDest, int pktSize);`
Inputs: • Packet type: int pktType (DATA = 0; ACK = 1) • Packet destination : int pktDest • Packet size: int pktSize in Bytes
Return value: SUCCESS = 1; FAIL = 0
Description: Create a packet to the destination address with assigned packet size and packet type with a unique sequence number.

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

...

Questions?

Backup slides

Implementation Details on WARP

User MAC code

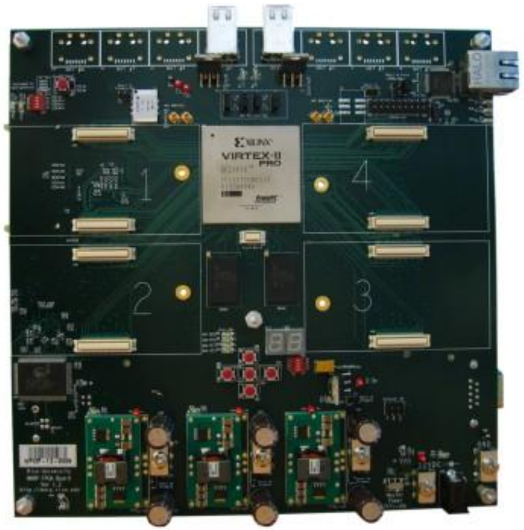
Framework object interface

Virtual component code	Object state information	Hardware scheduler

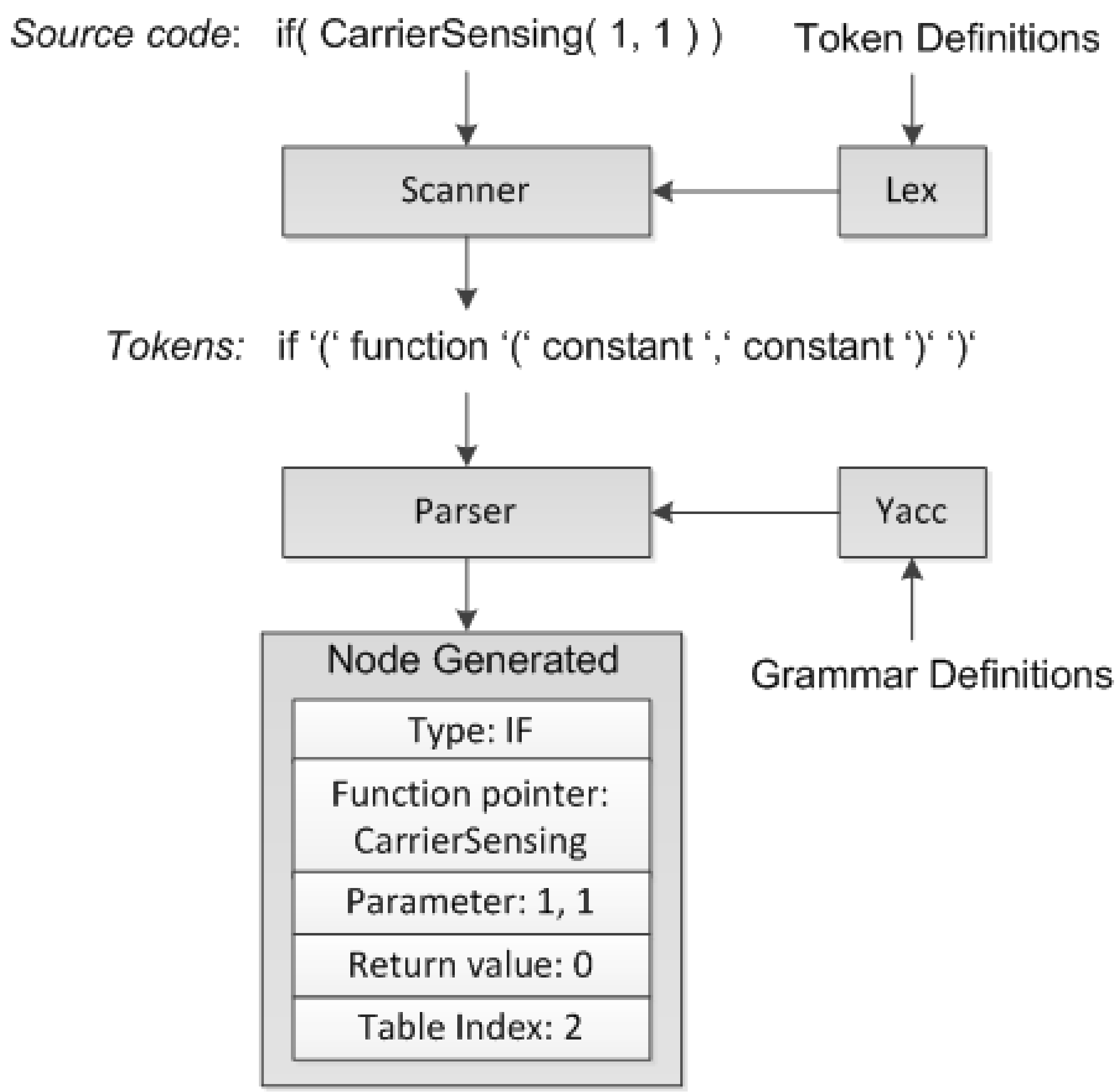
Platform-specific drivers

Hardware

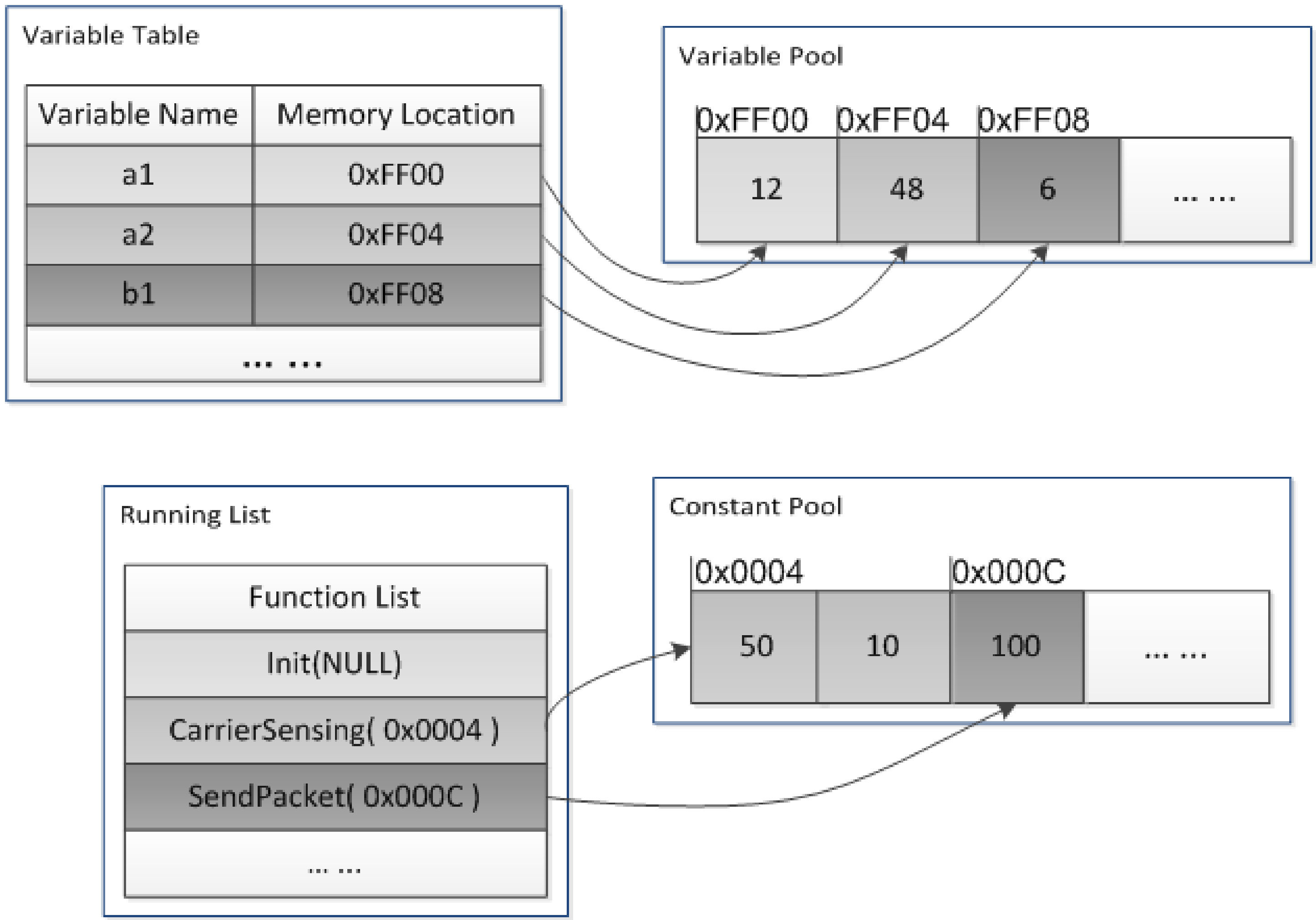
© http://warp.rice.edu/trac



Meta-compiler -cntd.-



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.