IEEE DySPAN'11 WARP Tutorial – Part II

A Compiler Assisted Approach for MAC Protocol Realization

Institute for Networked Systems,
RWTH Aachen University

1. Introduction

This section of the tutorial intends to give user hands-on-experience of TRUMP – a compiler assisted tool-chain for fast MAC protocol realization on WARP. We will be using a “drag and drop” Graphical User Interface (GUI) to auto-generate the corresponding MAC meta-language code which is sent to the WARP board over the UART interface. The code is then translated by a compiler running on the host WARP board to realize the corresponding MAC protocol. The flow of the tool-chain is shown in Figure. 1.

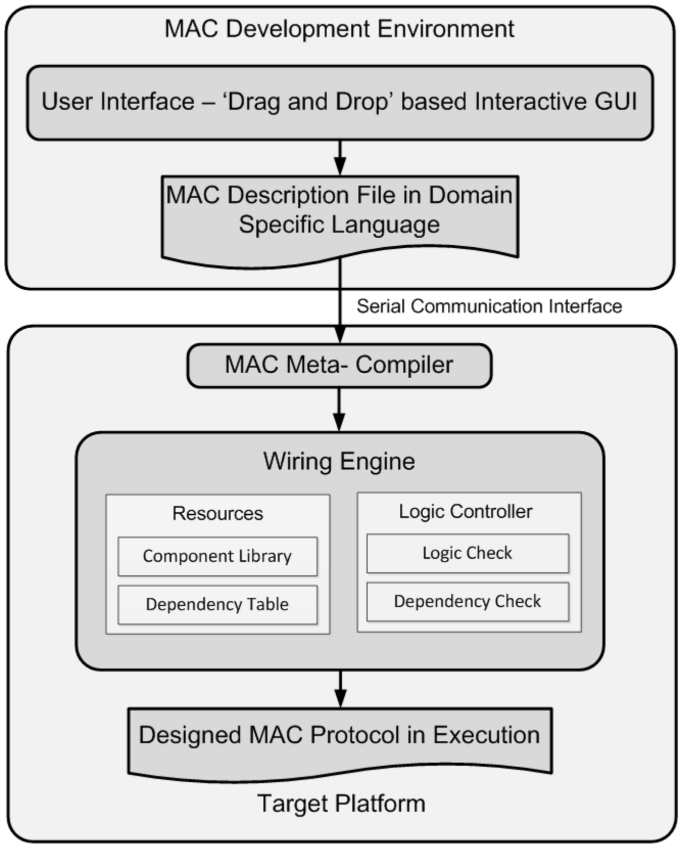

MAC Development Environment
User Interface – ‘Drag and Drop’ based Interactive GUI
MAC Description File in Domain Specific Language
Serial Communication Interface
MAC Meta- Compiler
Wiring Engine
Resources
Component Library
Dependency Table
Logic Controller
Logic Check
Dependency Check
Designed MAC Protocol in Execution
Target Platform

Figure 1: Design and execution flow of TRUMP

In our framework, a library of basic MAC components is provided through the GUI. These MAC components are identified by analyzing different types of MAC protocols and serve as the fundamental MAC building blocks. Our framework is implemented based on the OFDM Reference Design v14 on WARP v1 board.

In this tutorial session, we will implement a spectrum agile MAC protocol based on the multi-channel preamble reservation scheme using the GUI. For simplicity, we have divided the state diagram of the MAC protocol into transmission and reception parts as shown in Figure 2 and Figure 3, respectively.

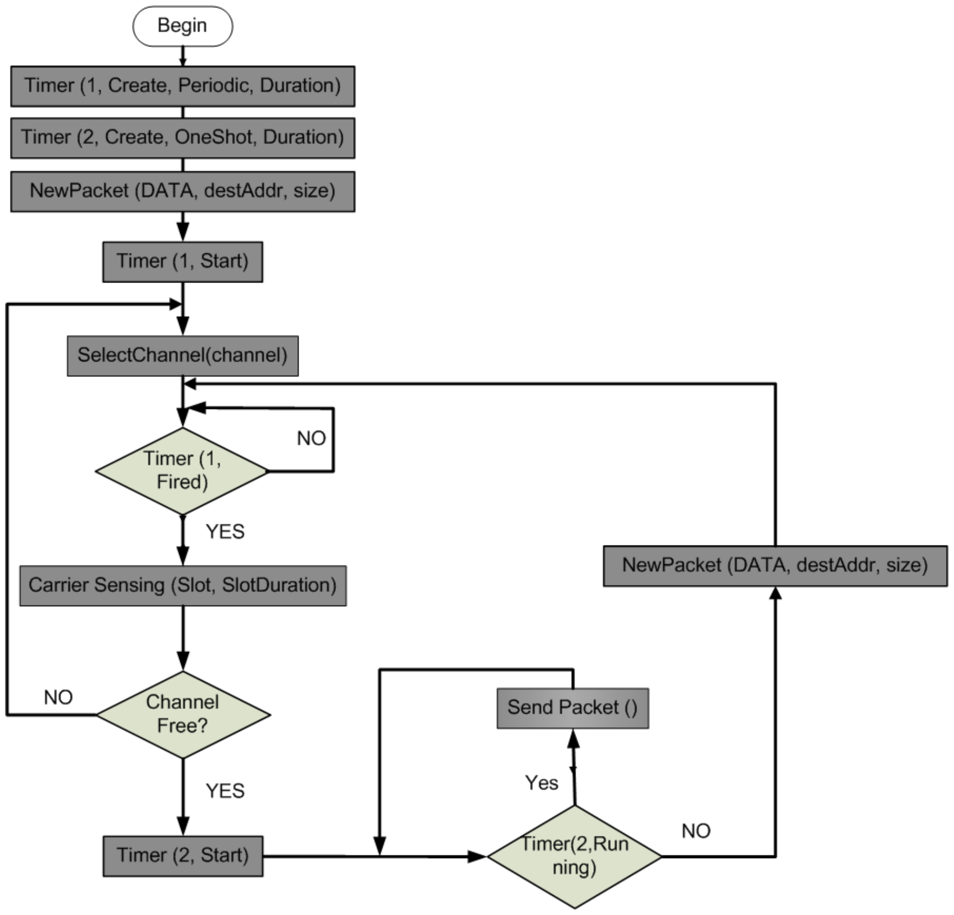

Begin
Timer (1, Create, Periodic, Duration)
Timer (2, Create, OneShot, Duration)
NewPacket (DATA, destAddr, size)
Timer (1, Start)
SelectChannel(channel)
Timer (1, Fired)
NO
YES
Carrier Sensing (Slot, SlotDuration)
Channel Free?
NO
YES
Timer (2, Start)
Timer(2,Running)
Yes
Send Packet ()
NO
NewPacket (DATA, destAddr, size)

Figure 2. The transmitter state diagram a simple spectrum agile MAC.

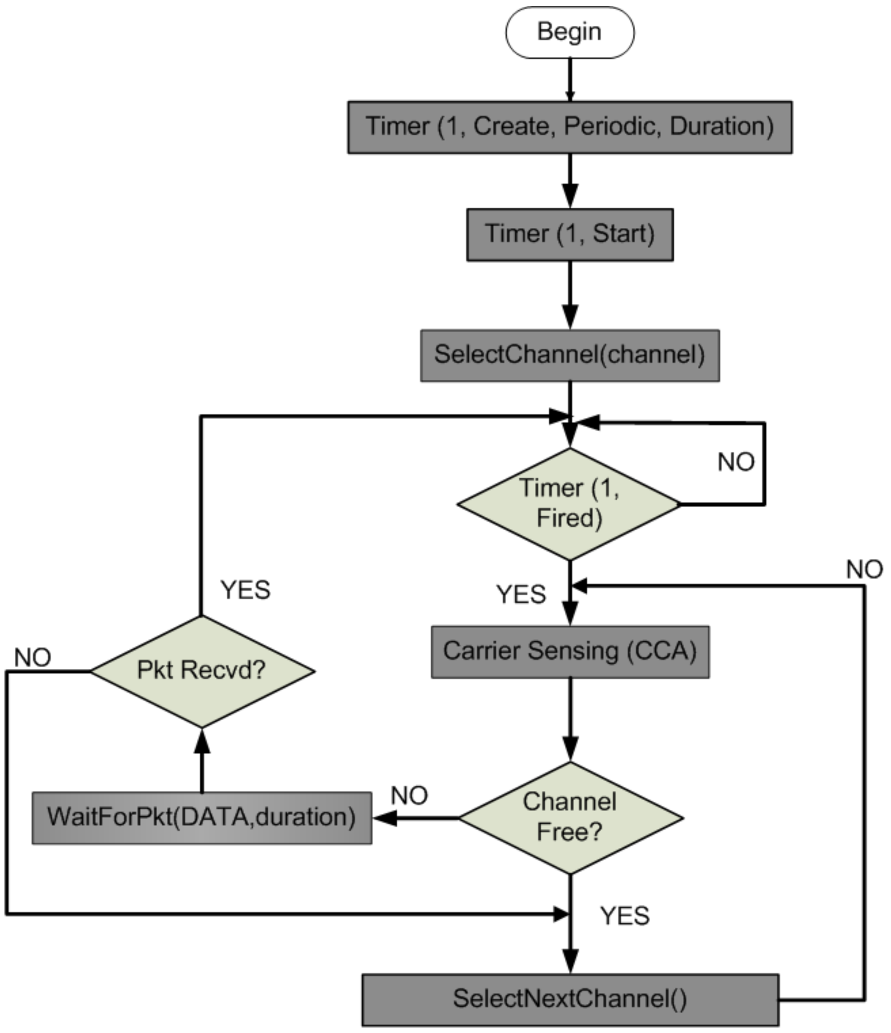
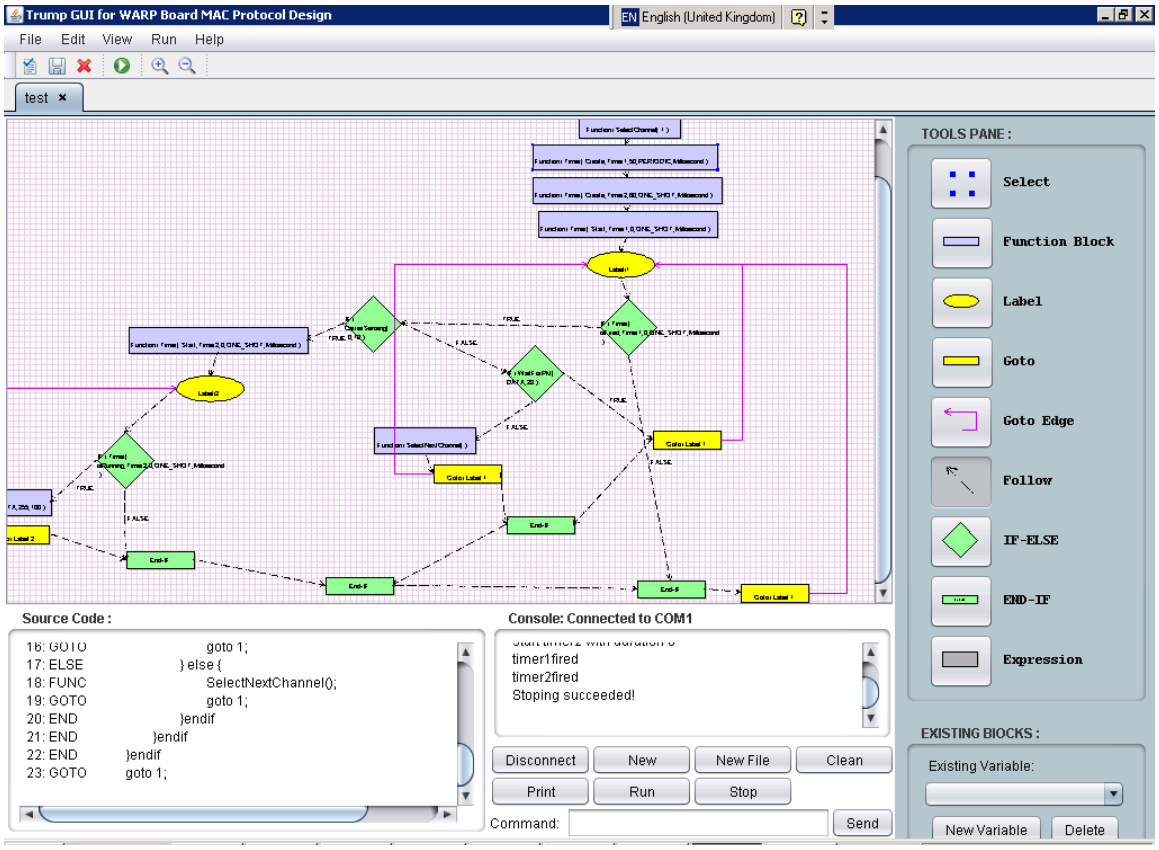


Figure 3: The receiver state diagram of simple spectrum agile MAC.

2. Instructions

1. Open the GUI interface.



```
GOTO                          goto 1;
ELSE                     } else {
FUNC                          SelectNextChannel();
GOTO                          goto 1;
END                      }endif
END                 }endif
END            }endif
GOTO           goto 1;
```

```
timer1fired
timer2fired
Stoping succeeded!
```

2. Select “for warp”.
3. Select “new block diagram”.
4. Create a transmitter followed by a receiver based on Figures 2 & 3 using the designated channels as shown at your workstation. The description of each library component and the parameters associated is listed in Appendix A.
5. Download your file to the WARP board using the “New” button.
6. One may also start/stop the MAC execution through the GUI.
7. One may use the function “ReportTxRxPkt” in the program to plot the transmission and reception throughput. To start plotting, one needs to 1) start tests.sln from folder Matlab_cpp_warp_interface, 2) Execute Matlab script TcpConnToWARP('127.0.0.1','1300')

3. Testing your MAC

One of the WARP boards is connected to a central server. It runs the same MAC protocol as the one to be implemented in the tutorial. One may test the implemented transmitter & receiver functionality by communicating with the central board.

A signal generator is used to act as an interferer. One should be able to observe that the MAC protocol is able to choose a non-jammed channel and establish a transmitter-receiver link.

One can observe the difference of throughput by scanning different number of channels, changing the packet size, changing carrier sensing duration, etc.

Appendix A:

TRUMP MAC WARP Board Library API for IEEE DySPAN Tutorial, May 2011.

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.`

Packet Reception
Functional API: `int WaitForPkt(int pktType, int waitDuration);`
Inputs: • Packet type: int pktType (DATA = 0; ACK = 1) • Duration for packet waiting: int waitDuration in Milliseconds
Return value: SUCCESS = 1 (Packet received); FAIL = 0
Description: This function switches the radio to listening mode. A timer of waitDuration is started at the beginning of the function. If no packet of the expected pktType is received during the waitDuration, the timer times out and the function exits with return value FAIL.

Timer

Functional API: `int Timer(int action, int timerID, duration, type, precision);`

Inputs:

- Timer action: int action
    - Create = 0: create a timer with duration, type, precision
    - Start = 1: start a timer that has been created. The duration of the timer can be reset by stop and start the timer again.
    - Stop = 2: stop a running timer.
    - isRunning = 3: checks if the timer is running
    - isFired = 4: wait for the timer to be fired
    - Destroy = 5: By default all the timers are destroyed at the end of the MAC execution. One can also voluntarily destroy a created timer if the timer is no longer in use.
- TimerID: 1-8
- Timer duration: in either Microseconds or Milliseconds at indicated by precision
- Timer type: One_shot = 0, Periodic = 1;
- Timer precision: Millisecond=0, Microsecond = 1.

Return value: SUCCESS = 1; FAIL = 0;

Description: This function is used to assign tasks to a timer of TimerID. All parameters are required when creating the timer, while only TimerID and action are required for other operations.

Set a set of channels to be used by the protocol

Functional API: `int SetChannelPool (int cha1, int cha2, int cha3, int cha4);`

Inputs: channel numbers: cha1, cha2, cha3, cha4.

Return value: SUCCESS = 1; FAIL = 0;

Description: This function is used to select the channels in 5GHz band to be used by the protocol. Up to four channels can be selected. If not indicated, the default values for the parameters are zero.

Select the next available channel in the channel pool
Functional API: `int SelectNextChannel();`
Inputs: None
Return value: SUCCESS = 1; FAIL = 0
Description: This function switches the current frequency band to the next available in the channel pool set by function SetChannelPool(). If channel pool is not set before the execution of this function, function returns FAIL.

Selecting a frequency channel on a WARP board in 5GHz band
Functional API: `void SelectChannel(unsigned char channel);`
Inputs: The channel to be used, `channel` of type unsigned char. The specified channels in 5GHz are: 1. 5180MHz 2. 5200MHz 3. 5220MHz 4. 5240MHz 5. 5260MHz 6. 5280MHz 7. 5300MHz 8. 5320MHz 9. 5500MHz 10. 5520MHz 11. 5540MHz 12. 5560MHz 13. 5580MHz 14. 5600MHz 15. 5620MHz 16. 5640MHz 17. 5660MHz 18. 5680MHz

19. 5700MHz 20. 5745MHz 21. 5765MHz 22. 5785MHz 23. 5805MHz
Return value: none
Description: This function is used to specify the frequency channel to be used on a WARP board. Before calling this function, the desired frequency band is selected.

Perform carrier sensing
Functional API: `int CarrierSensing(int num, int slot);`
Inputs: • num: number of times, the carrier is to be sensed • slot: value in microseconds and specifies the gap after which the carrier is sensed
Return value: 1 if the medium is free and 0 if the medium is false
Description: This function allows performing carrier sensing in a customized way with controllable gap interval and number of carrier sensing sampling.

Report current statistics on packet transmission and packet reception
Functional API: `int ReportTxRxPkt();`
Inputs: `None`
Return value: SUCCESS = 1; FAIL = 0;
Description: Pushes statics on packets transmitted and received in terms of bit per second to the Ethernet port.