/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* (C) Copyright 2000-2003, Wintegra. All rights reserved.

\* WINTEGRA CONFIDENTIAL PROPRIETARY

\* Contains Confidential Proprietary information of Wintegra.

\* Reverse engineering is prohibited.

\* The copyright notice does not imply publication.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Description:

\*

\* Bridging System (bridge1) between ENET ---> HDLC

\* PPPSwitching + Bridge System (bridge2) between HDLC and ENET

\*

\* Traffic

\*

\* Bridge1:

\* Host sends a packet on ENET1 (port is in internal loopback mode)

\* The packet is received into Bridge1. A DFC classifier rule based on INPUT\_IW\_PORT, ETHER\_TYPE and VLAN is applied that forwards the PACKET to one of the TDM ports (TDM #1 in this example).

\* TDM 1 was arbitarily picked, it could have been any TDM.

\*

\* Note that the flow aggregation that forwards the packet on TDM1, strips off the Ethernet header and prefixes a 4 byte HDLC header to the packet before sending it out on the TDM (hdlc\_prefix\_header)

\* If you want to change the hdlc\_prefix\_header, you will have to recompile the test.

\*

\* The TDM port is in loopback mode so the packet is received back into the PPP Switching System.

\*

\* PPP Switch:

\* NOTE: Address and Protocol Field compression are enabled on the PPP SWITCHING RX BINDING. So a packet that starts with the PPP header will have AFC/PFC applied to it.

\* Any packet that does not have a PPP header (Cisco HDLC packet) will assumed to have it compressed and will be accepted as is.

\* This is the only way to accept CISCO HDLC packets into the PPP Switching since they don't have a PPP header (0xFF03)

\*

\* So 0xFF03C021 is received back into the PPP Switching system as 0xC021 after applying AFC and PFC

\* So 0xFF030021 is received back into the PPP Switching system as 0x21 after applying AFC and PFC

\* So 0x0F008035 is received back into the PPP Switching system as 0x0F008035 since no PPP header is present (0xFF03)

\* So 0x0F000800 is received back into the PPP Switching system as 0x0F000800 since no PPP header is present (0xFF03)

\*

\* The PPP SWITCHING default flow aggregation (1 per TDM port) adds a dummy 14 byte ethernet header and forwards the packet to Bridge2

\*

\* Bridge2:

\* In Bridge2 we 3 user defined filters, that look for a user defined field starting at the ETHER\_TYPE reference point of the packet.

\*

\* The three filters (Filter0, Filter1, Filter2) are used for recognizing a 4 byte,2 byte and 1 byte header respectively (e.g 0x0F008035, 0xC021, 0x21)

\*

\* Two DFC rules are added for Filter0:

\* (1) For CISCO HDLC LCP packets (0x0F008035). This is sent to a flow agg named h\_flow\_agg\_cisco\_control and host terminated

\* (2) For CISCO HDLC IP DATA packets (0x0F000800). This is sent to a flow agg named h\_flow\_agg\_enet2 and sent out of ENET2

\*

\* One DFC rule is added for Filter1:

\* (1) For PPP LCP packets (0xc021). This is sent to a flow agg named h\_flow\_agg\_lcp and host terminated

\*

\* One rule is added for Filter2:

\* (1) For IP DATA packets (0x21). This is sent to the flow agg named h\_flow\_agg\_enet2 and sent out on ENET2

\*

\* Since ENET2 is in loopback mode, all packets sent out will be received back on ENET2 RX channel

\*

\* Finally look at App\_Debug that toggles the debug mode for Bridge1 to ether host terminate packets or forward packets on on the TDM

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define APP\_DEBUG 0

#define APP\_PPPSW 1

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include "wp\_wddi.h"

#include "wp\_sim.h"

#include "wpx\_oc12\_pmc5357.h"

#include "wt\_util.h"

#include "wpx\_pin\_mux\_selector.h"

#include "wpx\_tdm\_comet.h"

#include "wpx\_enet\_phy.h"

#define APP\_VLAN\_ID 0xa

#define APP\_MAX\_MAC\_ADDRESS 10

/\* Enet Bridge ports \*/

#define APP\_MAX\_BPORTS 20

#define APP\_MAX\_TDM\_PORTS 8

#define APP\_NUM\_IW\_BUFFERS 500

#define APP\_IW\_BUFFER\_SIZE 1984

#define APP\_IW\_BUFFER\_OFFSET 64

#define APP\_MASTER\_WPID WP\_WINPATH(0)

/\* Utility macros \*/

#define MAC\_COPY(d,s) { int \_i; \

for(\_i=0;\_i<6;\_i++) \

\*((d)+\_i)=\*((s)+\_i); }

extern WP\_U32 dps\_ProgramImage[];

extern WP\_U16 dps\_PC\_Table\_Init[];

WP\_context context\_cfg =

{

3, /\* qnodes \*/

/\* the first four assignments are mandatory bus assignments \*/

{

{/\* WP\_BUS\_PARAM \*/ WP\_BUS\_PARAM, APP\_BANK\_PARAM},

{/\* WP\_BUS\_PACKET \*/ WP\_BUS\_PACKET, APP\_BANK\_PACKET},

{/\* WP\_BUS\_INTERNAL \*/ WP\_BUS\_INTERNAL, APP\_BANK\_INTERNAL},

{/\* WP\_BUS\_HOST \*/ WP\_BUS\_HOST, APP\_BANK\_HOST},

{/\* WP\_BUS\_IRQ \*/ WP\_BUS\_HOST, APP\_BANK\_HOST},

{/\* WP\_BUS\_TMSTMP \*/ WP\_BUS\_HOST, APP\_BANK\_HOST},

{/\* WP\_VB\_CAS \*/ WP\_BUS\_HOST, APP\_BANK\_HOST},

{/\* UNASSIGNED \*/ WP\_BUS\_NONE, 0},

{/\* ASU\_TPT \*/ WP\_BUS\_PARAM, APP\_BANK\_PARAM},

{/\* ASU CW \*/ WP\_BUS\_INTERNAL, APP\_BANK\_INTERNAL},

{/\* ASU TX QUEUE \*/ WP\_BUS\_PARAM, APP\_BANK\_PARAM},

{/\* STATISTICS \*/ WP\_BUS\_HOST, APP\_BANK\_HOST},

{/\* ADDRESS LOOKUP \*/ WP\_BUS\_PARAM, APP\_BANK\_PARAM},

{/\* ATM CPT \*/ WP\_BUS\_PARAM, APP\_BANK\_PARAM},

{/\* ENET HDLC CPT \*/ WP\_BUS\_PARAM, APP\_BANK\_PARAM},

{/\* IW FBP PARAM \*/ WP\_BUS\_PARAM, APP\_BANK\_PARAM},

{/\* IW FBP DATA \*/ WP\_BUS\_PACKET, APP\_BANK\_PACKET},

{/\* EXTERNAL FIFO \*/ WP\_BUS\_PACKET, APP\_BANK\_PACKET},

{/\* MAC LOOKUP \*/ WP\_BUS\_PARAM, APP\_BANK\_PARAM},

{/\* IMA SMALL \*/ WP\_BUS\_INTERNAL, APP\_BANK\_INTERNAL},

{/\* IMA LARGE \*/ WP\_BUS\_HOST, APP\_BANK\_HOST},

{/\* LPM \*/ WP\_BUS\_HOST, APP\_BANK\_HOST},

{/\* MFC RULES \*/ WP\_BUS\_HOST, APP\_BANK\_HOST},

{/\* FIWT, QAT \*/ WP\_BUS\_PARAM, APP\_BANK\_PARAM},

{/\* MFCPT, UPPPT, DS\*/ WP\_BUS\_INTERNAL, APP\_BANK\_INTERNAL},

{/\* STATISTICS2 \*/ WP\_BUS\_PACKET, APP\_BANK\_PACKET},

{/\* STATISTICS3 \*/ WP\_BUS\_PACKET, APP\_BANK\_PACKET},

{/\* UNASSIGNED \*/ WP\_BUS\_NONE, 0},

{/\* WP\_VB\_PKT\_GTCPT \*/ WP\_BUS\_INTERNAL, APP\_BANK\_INTERNAL},

{/\* UNASSIGNED \*/ WP\_BUS\_NONE, 0},

{/\* MANDATORY \*/ WP\_BUS\_NONE, 0},

{/\* MANDATORY \*/ WP\_BUS\_NONE, 0},

{/\* WP\_VB\_MCH\_DATA \*/ WP\_BUS\_PACKET, APP\_BANK\_PACKET},

#if 1

{/\* WP\_VB\_OAM\_FM\_QUERY\*/ WP\_BUS\_NONE, 0},

{/\* WP\_VB\_HASH\_ON\_HEAP\*/ WP\_BUS\_NONE, 0},

{/\* WP\_VB\_MFC\_INTERNAL\*/ WP\_BUS\_PACKET, APP\_BANK\_PACKET}

#endif

},

{

dps\_ProgramImage, dps\_ProgramImage

},

{

dps\_PC\_Table\_Init, dps\_PC\_Table\_Init

},

NULL, /\* Interrupt queue tables \*/

0, /\* Mixumum number of pools \*/

NULL, /\* ATM global parameters \*/

NULL, /\* Interworking global parameters \*/

};

WP\_int\_queue\_table int\_queue\_table\_cfg =

{

{ /\* queue[WP\_IRQ\_TABLES] \*/

{2000, 1}, /\* size, rate \*/

{2000, 1},

{2000, 1},

{2000, 1}

}

};

WP\_pool\_ring\_data pool\_ring\_data\_cfg =

{

/\* n\_rings \*/ 50,

/\* ring\_length \*/ 30,

/\* bus \*/ WP\_BUS\_PARAM,

/\* bank \*/ APP\_BANK\_PARAM

};

WP\_qnode\_hostq qnode\_hostq\_cfg =

{

/\* pool\_buffer \*/ 0,

/\* pool\_ring \*/ 0,

/\* interruptqueue \*/ WP\_IRQT1

};

WP\_pool\_buffer\_data pool\_buffer\_data\_256\_cfg =

{

/\* n\_buffers \*/ 1000,

/\* offset \*/ 0,

/\* size \*/ 256,

/\* pad \*/ 0,

/\* bus \*/ WP\_BUS\_PACKET,

/\* bank \*/ APP\_BANK\_PACKET,

};

WP\_pool\_buffer\_data pool\_buffer\_data\_iw\_cfg =

{

/\* n\_buffers \*/ APP\_NUM\_IW\_BUFFERS,

/\* offset \*/ 64,

/\* size \*/ 1984,

/\* pad \*/ 0,

/\* bus \*/ WP\_BUS\_PACKET,

/\* bank \*/ APP\_BANK\_PACKET

};

WP\_atm\_global atm\_global\_cfg =

{

/\* max\_pm\_tables \*/ 0, /\* Performance monitoring \*/

/\* max\_ubrplus\_channels \*/ 0, /\* UBR+ channels configured \*/

/\* max\_upc\_tables \*/ 0, /\* Max policing tables \*/

/\* max\_rx\_cids \*/ 0, /\* AAL2 max Rx CIDs/channel \*/

/\* max\_tx\_cids \*/ 0, /\* AAL2 max Tx CIDs/channel \*/

/\* max\_cid\_range \*/ WP\_AAL2\_CIDRANGE\_UNUSED,

/\* qsr\_bus \*/ WP\_BUS\_NONE, /\* AAL2 QSR bus \*/

/\* qsr\_bank \*/ 0, /\* AAL2 QSR bus \*/

/\* atm\_modes \*/ /\* Atm modes \*/

{

WP\_ATM\_RAS\_TIMESTAMP\_ENABLE,

#if \_WP\_MODULE\_ABR\_ENABLE\_

0,

#endif

WP\_AAL2\_RX\_3FE\_DISABLE

},

/\* max\_stats\_extensions \*/ 0,

};

WP\_tdm\_slotgroup tdm\_slots[] =

{

{1, 24},

{0, 0}

};

/\* TDM set up \*/

WP\_port\_tdm port\_tdm\_cfg =

{

/\* atm\_limits\*/ {0},

/\* pkt\_limits\*/ {64,64,0,0},

/\* trans\_limits \*/ {{0},{0}},

/\* slot \*/ tdm\_slots,

/\* tdmmode \*/ WP\_TDM\_NORMAL,

/\* framemode \*/ WP\_TDM\_FRAMED,

/\* pinconfig \*/ WP\_TDM\_COMMON,

/\* edge \*/ WP\_TDM\_RISING,

/\* delay \*/ WP\_TDM\_DELAY\_1,

/\* intmode \*/ WP\_TDM\_INT\_DISABLE,

/\* rx\_iw\_bkgnd \*/ WP\_IW\_BKGND\_NOT\_USED

};

WP\_tdm\_binding hdlc\_binding[] =

{

{0, 23}

};

WP\_device\_tdm\_hdlc device\_tdm\_hdlc\_cfg =

{

/\* n\_timeslot\_binding \*/ 1,

/\* timeslot\_binding \*/ hdlc\_binding,

/\* tx\_statmode \*/ WP\_PKT\_STAT\_ENABLE,

/\* tx\_maxsdu \*/ 1536,

/\* rx\_statmode \*/ WP\_PKT\_STAT\_ENABLE,

/\* crctype \*/ WP\_HDLC\_CRC16,

/\* numflags \*/ 2,

/\* idlemode \*/ WP\_TDI\_IDLEMODE\_FLAG,

/\* flowmode \*/ WP\_FLOWMODE\_MULTI,

/\* addr\_mask \*/ 0,

/\* address1 \*/ 0,

/\* address2 \*/ 0,

/\* tx\_tqtypes \*/ { WP\_PKT\_SCHED\_WFQ,

WP\_PKT\_SCHED\_WFQ,

WP\_PKT\_SCHED\_WFQ,

WP\_PKT\_SCHED\_WFQ

}

};

WP\_port\_enet port\_enet\_cfg =

{

/\* pkt\_limits \*/ {2,2},

/\* flowmode \*/ WP\_FLOWMODE\_FAST,

/\* miimode \*/ WP\_ENET\_RGMII\_1000,

/\* rx\_iw\_bkgnd \*/ WP\_IW\_BKGND\_USED,

};

WP\_enet\_filters enet\_filters\_cfg =

{

/\*rx\_ucfilter;\*/WP\_ENET\_FILTER\_PROCESS,

/\*rx\_bcfilter;\*/WP\_ENET\_FILTER\_HOSTTERM,

/\*rx\_mcfilter;\*/WP\_ENET\_FILTER\_HOSTTERM,

};

WP\_device\_enet device\_enet\_cfg =

{

/\* max\_tx\_channels \*/ 1,

/\* tx\_maxsdu \*/ 1536,

/\* rmii\_operating\_speed \*/ WP\_UNUSED,

/\* mac\_addr[6] \*/ {0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa},

/\* tx\_bit\_rate \*/ 1000000000,

/\* loopbackmode \*/ WP\_ENET\_LOOPBACK,

/\* extended\_params \*/ NULL,

};

WP\_pkt\_shaping\_wfq default\_pkt\_shaping\_wfq\_cfg =

{

/\* weight; \*/1,

/\* weight\_fraction;\*/0

};

WP\_ch\_enet ch\_enet\_cfg =

{

/\* intmode \*/ WP\_PKTCH\_INT\_ENABLE,

/\* iwmmode \*/ WP\_PKTCH\_IWM\_ENABLE,

/\* testmode \*/ WP\_PKTCH\_TEST\_DISABLE,

/\* tx\_pqblock \*/ 0,

/\* tx\_pqlevel \*/ 0,

/\* tx\_shaping\_type \*/ WP\_PKT\_SHAPING\_STRICT,

/\* tx\_shaping\_params \*/ &default\_pkt\_shaping\_wfq\_cfg,

/\* rx\_maxsdu \*/ 1536,

/\* tx\_cwid \*/ WP\_CW\_ID\_A,

/\* tx\_tq \*/ 0,

/\* rx\_queuedepth \*/17,

};

WP\_pkt\_shaping\_cir pkt\_shaping\_cir\_cfg =

{

/\* cir \*/ 11000,

/\* polling \*/ WP\_SHAPING\_POLL\_AUTOD,

/\* cbs \*/ 0,

};

/\* Configuration structure for the HDLC channels\*/

WP\_ch\_hdlc ch\_hdlc\_cfg =

{

/\* intmode \*/ WP\_PKTCH\_INT\_DISABLE,

/\* iwmmode \*/ WP\_PKTCH\_IWM\_ENABLE,

/\* testmode \*/ WP\_PKTCH\_TEST\_DISABLE,

/\* tx\_pqblock \*/ 0,

/\* tx\_pqlevel \*/ 0,

/\* tx\_shaping\_type \*/ WP\_PKT\_SHAPING\_WFQ,

/\* tx\_shaping\_params \*/ &default\_pkt\_shaping\_wfq\_cfg,

/\* rx\_maxsdu \*/ 1536,

/\* tw\_cwid \*/ WP\_CW\_ID\_A,

/\* tx\_tq \*/ 0

};

WP\_calendar\_wheel cw\_config[2] =

{

{

/\* function \*/ WP\_CW\_PSU,

/\* cwid \*/ WP\_CW\_ID\_A,

/\* min\_ch\_speed \*/ 10000,

/\* max\_ch\_speed \*/ 30000

},

WP\_CALENDAR\_WHEEL\_TERM

};

WP\_iw\_global iw\_global\_cfg =

{

/\* max\_iw\_aggregations \*/ 200,

/\* max\_routing\_systems \*/ 0,

/\* max\_directmap\_ene... \*/ 1,

/\* max\_bridging\_systems \*/ 2,

/\* max\_mpls\_systems \*/ 0,

/\* max\_vlan\_priority... \*/ 0,

/\* iw\_bkgnd\_fifo\_size \*/ 1000,

/\* cong\_pt \*/ NULL,

/\* iw\_host\_limits \*/ {10, 10, 0,0 },

/\* mpls\_config \*/ {0, 0},

/\* iw\_modes \*/ {

WP\_IW\_POLICER\_ENABLE,

WP\_IW\_STATISTICS\_BUNDLE\_STAT\_ENABLE,

WP\_IW\_L2\_HEADER\_EXTENSION\_ENABLE,

WP\_IW\_ENHANCED\_FLOW\_STAT\_ENABLE,

WP\_IW\_FLOW\_STAT\_ENABLE,

WP\_IW\_FR\_TX\_FLOW\_STAT\_DISABLE,

0,

WP\_IW\_LEARNING\_DISABLE,

/\* port\_filtering\_mode; \*/ WP\_IW\_PORT\_FILTERING\_ENABLE,

/\* lpm\_alloc\_mode; \*/ 0,

/\* enhanced\_mc\_member\_stat\_mode; \*/ 0,

0,

1,

},

/\* max\_iw\_stat\_bundles \*/ 0,

/\* max\_fr\_sw\_systems \*/ 0,

/\* max\_bridging ports \*/ APP\_MAX\_BPORTS,

/\* max\_iw\_mc\_groups \*/ 0,

/\* max\_nat\_systems \*/ 0,

/\* max\_iw\_second\_agg \*/ 0,

/\* max\_iw\_compr\_limits \*/ 0,

/\* WP\_U32 max\_iw\_mc\_members \*/ 0,

/\* WP\_U16 max\_iw\_programmable\_filters \*/ 3,

/\* WP\_U16 max\_iw\_policers \*/ 1,

};

WP\_tx\_binding tx\_binding\_cfg =

{

/\* res0 \*/ 0,

/\* dci\_mode \*/ WP\_IW\_DYN\_CH\_INSERT\_ENABLE,

/\* maxt \*/ 10,

};

WP\_qnode\_iwq qnode\_iwq\_cfg =

{

/\* interruptqueue \*/ WP\_IRQT1,

/\* num\_buffers \*/ 1000,

/\* adjunct\_pool \*/ 0,

};

/\* INTERWORKING BRIDGING \*/

WP\_ch\_iw\_rx ch\_iw\_rx\_cfg =

{

/\* pqblock \*/ 0,

/\* pqlevel \*/ 0,

/\* tx\_binding\_type \*/ WP\_IW\_TX\_BINDING,

/\* tx\_binding\_config \*/ NULL,

};

WP\_rx\_binding\_bridging rx\_binding\_bridging\_cfg =

{

/\* encap\_mode \*/ WP\_IW\_VCMUX,

/\* mru \*/ 1536,

/\* vcfcs \*/ 0,

/\* input\_port \*/ 0,

};

WP\_iw\_agg\_bridging default\_iw\_agg\_bridging\_cfg =

{

/\* tag \*/ 0,

/\* txfunc \*/ 0,

/\* bport \*/ 0,

/\* rfcs \*/ WP\_IW\_RFCS\_ENABLE,

/\* l2\_header\_insert\_mode \*/ WP\_IW\_L2H\_INSERT\_DISABLE,

/\* vlan\_tag\_mode \*/ WP\_IW\_VLAN\_TAG\_DISABLE,

/\* interruptqueue \*/ WP\_IW\_IRQT1,

/\* error\_pkt\_mode \*/ WP\_IW\_ERRPKT\_DISCARD,

/\* intmode \*/ WP\_IW\_INT\_ENABLE,

/\* statmode \*/ WP\_IW\_STAT\_ENABLE,

/\* timestamp\_mode \*/ WP\_IW\_TIME\_STAMP\_DISABLE,

/\* ov\_pool\_mode \*/ WP\_IW\_OV\_POOL\_DISABLE,

/\* fbp\_drop\_threshold \*/ 0,

/\* replace\_vlan\_tag \*/ WP\_IW\_REPLACE\_VTAG\_DISABLE,

/\* vlan\_id \*/ 0,

/\* vpmt\_handle \*/ 0,

/\* mtu \*/ 1536,

/\* prefix\_length \*/ 0,

/\* prefix\_header[28] \*/ {0},

/\* policer\_enable \*/ WP\_IW\_POLICER\_DISABLE,

/\* policer\_config \*/ NULL,

/\* cong\_mode \*/ WP\_IW\_CONGESTION\_DISABLE,

/\* cong\_threshold\_param \*/ NULL ,

/\* flow\_agg\_type \*/ WP\_IW\_FLOW\_AGG\_PRIMARY ,

/\* external\_vlan \*/ {

WP\_IW\_EXTERN\_VLAN\_NONE,

0

},

/\* bc\_mc\_priority\_mapping \*/ {

WP\_IW\_BC\_PRIORITY\_DISABLE,

0,

WP\_IW\_MC\_PRIORITY\_DISABLE,

0

},

/\* reserved\_agg\_mode \*/ WP\_IW\_NOT\_RESERVED,

/\* next\_preserve\_inport \*/ WP\_IW\_OVERRIDE\_INPORT,

/\* next\_preserve\_inport\_params \*/ 0,

/\* mac\_replace\_mode \*/ WP\_IW\_MAC\_REPLACE\_DISABLED,

/\* new dst mac \*/ {0},

/\* new\_src\_mac \*/ {0},

/\* multi\_cong\_prof \*/ 0,

/\* extraction\_length \*/ 0,

};

WP\_dfc\_sys\_info dfc\_sys\_info =

{

/\* classification hash size \*/ WP\_IW\_32K\_HASH\_ENTRIES,

/\* port filtering \*/ WP\_IW\_PORT\_FILTERING\_ENABLE,

/\* deny\_eq\_ip \*/ WP\_IW\_DENY\_EQUAL\_IP\_DISABLE,

};

WP\_iw\_sys\_bridging iw\_sys\_bridge =

{

/\* tag \*/ 0,

/\* max\_flows \*/ 10,

/\* classification mode \*/ WP\_IW\_CLASSIFIER\_BASED\_BRIDGING,

/\* classifier config \*/ {WP\_IW\_DYNAMIC\_CLASS,100,0,NULL},

/\* learning\_mode \*/ WP\_IW\_BRIDGE\_LEARNING\_DISABLE,

/\* learning queue \*/ {

WP\_IW\_IRQT1,

0,

WP\_LEARNING\_INTENABLE,

1,

},

/\* forwarding\_table\_size \*/ WP\_IW\_32K\_HASH\_ENTRIES,

/\* member\_set\_size \*/ WP\_IW\_32K\_HASH\_ENTRIES,

/\* buffer\_gap \*/ 0x40,

/\* max\_bridging\_ports \*/ APP\_MAX\_BPORTS,

/\* dfc\_info \*/ &dfc\_sys\_info,

/\* svl\_mode \*/ WP\_IW\_SVL\_DISABLED,

};

WP\_bridge\_port bridge\_port\_cfg =

{

/\* tag \*/ 0,

/\* direct\_mapping \*/ WP\_IW\_DIRECT\_MAP\_DISABLE,

/\* flow\_agg \*/ 0,

/\* termination\_mode \*/ WP\_IW\_HOST\_TERM\_MODE,

/\* learning\_mode \*/ WP\_IW\_LEARNING\_DISABLE,

/\* in\_filter\_mode \*/ WP\_IW\_INGRESS\_FILTER\_DISABLE,

/\* vlan\_param \*/ {

WP\_IW\_ACCEPT\_TAGGED\_UNTAGGED,

0,

/\* vlan tunnel \*/ 0,

/\* vlan\_priority\_enforcement \*/ WP\_IW\_VLAN\_PRIORITY\_ENFORCE\_DISABLED,

},

/\* max\_mac\_addresses \*/ APP\_MAX\_MAC\_ADDRESS,

/\* group\_tag \*/ 0,

/\* group\_filtering\_mode \*/ WP\_IW\_GROUP\_FILTER\_DISABLE,

/\*unk\_mac\_sa\_filter;\*/0,

/\*unk\_mc\_mode;\*/0,

/\*bc\_ht\_mode;\*/0,

/\*input\_filters\_mask;\*/0,

/\*output\_filters\_mask;\*/0,

/\*statmode;\*/ WP\_IW\_PORT\_STAT\_ENABLE,

/\* unk\_uc\_mode;\*/0,

/\* classification\_flag;\*/ WP\_IW\_IWPORT\_CLASSIFICATION\_ENABLED,

/\* adv\_unk\_lookup\_mode;\*/ 0,

/\* cfi\_ht\_mode;\*/ 0,

/\* reserved\_mc\_ht\_mode;\*/ WP\_IW\_RES\_MC\_HT\_ENABLE,

};

WP\_iw\_sys\_directmap iw\_sys\_pppsw\_cfg =

{

/\* max\_flows \*/ APP\_MAX\_TDM\_PORTS,

/\* buffer\_gap \*/ 0x40

};

WP\_iw\_agg\_directmap iw\_agg\_directmap\_cfg =

{

/\* tag \*/ 0,

/\* txfunc \*/ 0,

/\* rfcs \*/ WP\_IW\_RFCS\_DISABLE,

/\* l2header\_insert\_mode \*/ WP\_IW\_L2H\_INSERT\_DISABLE,

/\* interruptqueue \*/ WP\_IW\_IRQT1,

/\* error\_pkt\_mode\*/ WP\_IW\_ERRPKT\_DISCARD,

/\* intmode \*/ WP\_IW\_INT\_DISABLE,

/\* statmode \*/ WP\_IW\_STAT\_ENABLE,

/\* timestamp\_mode \*/ WP\_IW\_TIME\_STAMP\_DISABLE,

/\* ov\_pool\_mode\*/ 0,

/\* fbp\_drop\_threshold \*/ 0,

/\* mtu \*/ 1536,

/\* prefix\_length \*/ 0,

/\* extraction\_length \*/ 0,

/\* prefix\_header[36] \*/ {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00},

/\* policer\_enable \*/ 0,

/\* policer\_config \*/ NULL,

/\* cong\_mode \*/ 0,

/\* cong\_threshold\_param \*/ NULL

};

WP\_iw\_agg\_ppp\_switching iw\_agg\_ppp\_switching =

{

/\* tag \*/ 0,

/\* txfunc \*/ 0,

/\* l2header\_insert\_mode \*/ WP\_IW\_L2H\_INSERT\_DISABLE,

/\* interruptqueue \*/ WP\_IW\_IRQT1,

/\* error\_pkt\_mode\*/ WP\_IW\_ERRPKT\_DISCARD,

/\* intmode \*/ WP\_IW\_INT\_DISABLE,

/\* statmode \*/ WP\_IW\_STAT\_ENABLE,

/\* timestamp\_mode \*/ WP\_IW\_TIME\_STAMP\_DISABLE,

/\* ov\_pool\_mode\*/ 0,

/\* fbp\_drop\_threshold \*/ 0,

/\* mtu \*/ 1536,

/\* prefix\_length \*/ 0,

/\* extraction\_length \*/ 0,

/\* prefix\_header[36] \*/ {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00},

/\* policer\_enable \*/ 0,

/\* policer\_config \*/ NULL,

/\* cong\_mode \*/ 0,

/\* cong\_threshold\_param \*/ NULL,

/\* flow agg type \*/ WP\_IW\_AGG\_PRIMARY,

/\* acfc\_mode \*/ WP\_IW\_ACFC\_ENABLE,

/\* pfc\_mode \*/ WP\_IW\_PFC\_ENABLE,

/\* policer handle \*/ 0,

/\* iwport \*/ 0,

};

/\*

\* Global variables used during test execution

\*/

WP\_U8 wt\_mac\_enet1[6] = {0x45, 0x6e, 0x65, 0x74, 0x23, 0x31};

WP\_U8 wt\_mac\_enet2[6] = {0x45, 0x6e, 0x65, 0x74, 0x23, 0x32};

WP\_tag tag\_channel\_base = 1;

WP\_flow\_learning\_rule\_vlan bridge\_cfg;

WP\_l2\_forward forward\_rule;

/\* Interrupt handling structures \*/

typedef struct {

WP\_tag event\_tag;

WP\_U32 event\_type;

void \*event\_param;

} app\_task;

typedef struct {

WP\_U16 head;

WP\_U16 tail;

WP\_U16 num\_elements;

WP\_U16 pad;

app\_task \*task;

} app\_task\_list;

/\* Interrupt task list \*/

#define IRQ\_TASK\_LIST\_SIZE 4096

app\_task irq\_task[IRQ\_TASK\_LIST\_SIZE];

app\_task\_list irq\_task\_list[1] = { { 0, 0, IRQ\_TASK\_LIST\_SIZE, 0, irq\_task} };

/\* Application prototypes \*/

void App\_Quit(void);

void App\_EventRxIndicate(WP\_tag tag, WP\_U32 info, WP\_U32 data);

void App\_EventTxIndicate(WP\_tag tag, WP\_U32 info, WP\_U32 data);

void App\_EventErrorIndicate(WP\_tag tag, WP\_U32 info, WP\_U32 data);

void App\_ShowStats(void);

void App\_Debug(WP\_boolean debug);

void App\_TerminateOnError(WP\_handle handle, WP\_CHAR \*s);

void App\_CreateEnetHdlcBridge(void);

void App\_CreateHdlcEnetBridge(void);

void App\_DuReceive(WP\_handle h\_rx, WP\_U32 data\_type);

void App\_DataSend(WP\_handle h\_dev, WP\_handle h\_pool);

void WPI\_SimulateInterrupts(void);

app\_task \*next\_task(app\_task\_list \*task\_list, app\_task \*result);

void add\_task(app\_task\_list \*task\_list, WP\_U32 event\_type, WP\_tag tag,

void \*event\_param);

void app\_perform\_action(app\_task \*task);

#define ENET\_HEADER\_SIZE 14

#define IP\_HEADER\_SIZE 20

/\* Byte offsets from start of IP header \*/

#define IP\_VER\_HLEN 0

#define IP\_SER\_TYPE 1

#define IP\_LENGTH 2

#define IP\_TTL 8

#define IP\_PROTOCOL 9

#define IP\_CHECKSUM 10

#define IP\_SA 12

#define IP\_DA 16

#define UDP\_HEADER\_SIZE 8

#define UDP\_SRC\_PORT 0

#define UDP\_DST\_PORT 2

#define UDP\_LENGTH 4

#define UDP\_CHECKSUM 6

/\* Functions provided by wt\_util.c. \*/

void insert\_ip\_header(WP\_U32 ip\_sa, WP\_U32 ip\_da, WP\_U32 payload\_size,

WP\_U32 protocol, WP\_U32 length, WP\_U8 \*prefix);

/\* ETHERNET \*/

WP\_handle h\_port\_enet1, h\_port\_enet2;

WP\_handle h\_device\_enet1, h\_device\_enet2;

WP\_handle h\_port\_tdm[APP\_MAX\_TDM\_PORTS];

WP\_handle h\_dev\_tdm[APP\_MAX\_TDM\_PORTS];

/\* INTERWORKING COMMON \*/

WP\_handle h\_pool\_buffer\_host;

WP\_handle h\_pool\_ring\_host;

WP\_handle h\_pool\_buffer\_iw,h\_pool\_256;

WP\_handle h\_qnode\_iwq, h\_qnode\_host,h\_qnode\_iwq\_tdm;

WP\_handle h\_bridge1\_iwhost, h\_bridge2\_iwhost;

WP\_handle h\_bridge1\_default\_agg,h\_bridge2\_default\_agg;

WP\_handle h\_flow\_agg\_tdm[APP\_MAX\_TDM\_PORTS];

WP\_handle h\_iw\_sys\_bridge1, h\_iw\_sys\_bridge2;

WP\_handle h\_iw\_sys\_pppsw;

WP\_handle h\_learningflow\_enet1;

WP\_handle h\_bridge1\_dfcflow1;

WP\_handle h\_bridge2\_dfcflow[4];

WP\_handle h\_bridge1\_filter1,h\_bridge2\_filter[3];

WP\_handle h\_port\_iwhost,h\_dev\_iwhost;

WP\_handle h\_bridge1\_bport\_enet1, h\_bridge1\_bport\_tdm[APP\_MAX\_TDM\_PORTS];

WP\_handle h\_bridge2\_bport\_enet2, h\_bridge2\_bport\_tdm[APP\_MAX\_TDM\_PORTS];

WP\_handle h\_flow\_agg\_pppsw\_link[APP\_MAX\_TDM\_PORTS];

WP\_handle h\_flow\_agg\_lcp, h\_flow\_agg\_cisco\_control,h\_flow\_agg\_enet2;

WP\_handle h\_rport\_tdm[APP\_MAX\_TDM\_PORTS], h\_rport\_enet2;

WP\_handle h\_enet1\_rx, h\_enet1\_host\_tx, h\_enet1\_tx;

WP\_handle h\_enet2\_rx, h\_enet2\_tx;

WP\_handle h\_tdm\_rx[APP\_MAX\_TDM\_PORTS],h\_tdm\_tx[APP\_MAX\_TDM\_PORTS];

WP\_handle status;

/\* TAGS \*/

WP\_tag tag\_agg\_enet;

WP\_tag tag\_bport\_enet1;

WP\_tag tag\_bport\_enet2;

WP\_tag tag\_bridge1\_bport\_tdm[APP\_MAX\_TDM\_PORTS];

WP\_tag tag\_bridge2\_bport\_tdm[APP\_MAX\_TDM\_PORTS];

WP\_tag tag\_rport\_tdm[APP\_MAX\_TDM\_PORTS];

WP\_tag tag\_rport\_enet2;

WP\_tag tag\_agg\_enet2;

WP\_tag tag\_iw\_sys\_bridge1, tag\_iw\_sys\_bridge2;

WP\_tag tag\_agg\_default\_bridge1, tag\_agg\_default\_bridge2;

WP\_tag tag\_enet1\_rx, tag\_enet2\_rx;

WP\_tag tag\_enet1\_tx, tag\_enet2\_tx;

WP\_tag tag\_tdmrx = 200;

WP\_tag tag\_tdmtx = 300;

WP\_tag tag\_agg\_tdm[APP\_MAX\_TDM\_PORTS];

WP\_tag tag\_agg\_pppsw[APP\_MAX\_TDM\_PORTS];

WP\_tag tag\_agg\_lcp, tag\_agg\_cisco\_control;

WP\_boolean debug\_on = WP\_FALSE;

int main(void)

{

WP\_U32 ii;

WP\_status status;

app\_task \*task, a\_task;

WP\_CHAR comm;

WP\_U32 tag1,tag2;

tag\_agg\_enet= 500;

tag\_iw\_sys\_bridge1 = 600;

tag\_iw\_sys\_bridge2 = 601;

tag\_bport\_enet1 = 700;

tag\_bport\_enet2 = 701;

for(ii = 0;ii < APP\_MAX\_TDM\_PORTS;ii++)

{

tag\_bridge1\_bport\_tdm[ii] = 802 + ii;

tag\_bridge2\_bport\_tdm[ii] = 902 + ii;

}

tag\_rport\_enet2 = 1000;

for(ii = 0;ii < APP\_MAX\_TDM\_PORTS;ii++)

{

tag\_agg\_tdm[ii] = 1100 + ii;

tag\_agg\_pppsw[ii] = 1200 + ii;

}

tag\_agg\_lcp = 1250;

tag\_agg\_cisco\_control = 1251;

tag\_agg\_enet2 = 1300;

tag\_agg\_default\_bridge1 = 1400;

tag\_agg\_default\_bridge2 = 1500;

tag\_enet1\_rx = 1600;

tag\_enet1\_tx = 1700;

tag\_enet2\_rx = 1800;

tag\_enet2\_tx = 1900;

/\* Context configuration \*/

context\_cfg.int\_queue\_data = &int\_queue\_table\_cfg;

context\_cfg.atm\_data = &atm\_global\_cfg;

context\_cfg.iw\_data = &iw\_global\_cfg;

context\_cfg.pool\_limit = 8;

context\_cfg.max\_qnodes = 4;

status = WP\_DriverInit();

App\_TerminateOnError(status,"WP\_DriverInit()");

status = WP\_SysInit(APP\_MASTER\_WPID, &context\_cfg);

App\_TerminateOnError(status, "WP\_SysInit()");

status = WPU\_EnetPhyInit(WP\_PORT\_ENET1, WPU\_MODE\_RGMII | WPU\_OV\_FIBER);

App\_TerminateOnError(status, " WPU\_WinnetPhyInit - WP\_PORT\_ENET1");

status = WPU\_EnetPhyInit(WP\_PORT\_ENET3, WPU\_MODE\_RGMII | WPU\_OV\_FIBER);

App\_TerminateOnError(status, " WPU\_WinnetPhyInit - WP\_PORT\_ENET1");

/\* Create BRG1 & BRG3 \*/

status = WP\_SysClockCreate(WP\_WINPATH(0), WP\_BRG1, WP\_BRG\_SRC\_BRGIN2, 2);

status = WP\_SysClockCreate(WP\_WINPATH(0), WP\_BRG2, WP\_BRG\_SRC\_BRGIN2, 2);

status = WP\_SysClockCreate(WP\_WINPATH(0), WP\_BRG3, WP\_BRG\_SRC\_BRGIN2, 2);

status = WP\_SysClockCreate(WP\_WINPATH(0), WP\_BRG4, WP\_BRG\_SRC\_BRGIN2, 2);

status = WP\_SysClockCreate(WP\_WINPATH(0), WP\_BRG5, WP\_BRG\_SRC\_BRGIN2, 2);

/\* Create Enet1 port \*/

h\_port\_enet1 = WP\_PortCreate(APP\_MASTER\_WPID, WP\_PORT\_ENET1,

&port\_enet\_cfg);

App\_TerminateOnError(h\_port\_enet1, "WP\_PortCreate() Enet1");

/\* Create device on Enet1 \*/

MAC\_COPY(device\_enet\_cfg.mac\_addr, wt\_mac\_enet1);

h\_device\_enet1 = WP\_DeviceCreate(h\_port\_enet1, WP\_PHY(0), WP\_DEVICE\_ENET,

&device\_enet\_cfg);

App\_TerminateOnError(h\_device\_enet1, "WP\_DeviceCreate() Enet1");

status = WPU\_TdmCometCardInit(WP\_PORT\_TDM1, WPU\_INIT\_COMET\_CPLD\_T1);

App\_TerminateOnError(status, "WPU\_TdmCometCardInit");

/\* Create Enet2 port \*/

h\_port\_enet2 = WP\_PortCreate(APP\_MASTER\_WPID, WP\_PORT\_ENET3,

&port\_enet\_cfg);

App\_TerminateOnError(h\_port\_enet2, "WP\_PortCreate() Enet2");

/\* Create device on Enet2 \*/

MAC\_COPY(device\_enet\_cfg.mac\_addr, wt\_mac\_enet2);

h\_device\_enet2 = WP\_DeviceCreate(h\_port\_enet2, WP\_PHY(0), WP\_DEVICE\_ENET,

&device\_enet\_cfg);

App\_TerminateOnError(h\_device\_enet2, "WP\_DeviceCreate() Enet1");

/\* Create TDM ports and devices \*/

for(ii = 0;ii < APP\_MAX\_TDM\_PORTS;ii++)

{

h\_port\_tdm[ii] = WP\_PortCreate(WP\_WINPATH(0), WP\_PORT\_TDM1 + ii, &port\_tdm\_cfg);

App\_TerminateOnError(h\_port\_tdm[ii],"WP\_PortCreate() TDM1");

h\_dev\_tdm[ii] = WP\_DeviceCreate(h\_port\_tdm[ii], WP\_PHY(0), WP\_DEVICE\_HDLC,

&device\_tdm\_hdlc\_cfg);

App\_TerminateOnError(h\_dev\_tdm[ii],"WP\_DeviceCreate() TDM1");

status = WPU\_TdmCometCardInit(WP\_PORT\_TDM1 + ii, WPU\_MODE\_TDM\_T1\_MASTER),

App\_TerminateOnError(status, "WPU\_TdmCometCardInit()");

}

/\* Create an IW Host port \*/

h\_port\_iwhost = WP\_PortCreate(WP\_WINPATH(0), WP\_PORT\_IW\_HOST, NULL);

App\_TerminateOnError(h\_port\_iwhost, "WP\_PortCreate() IW Host");

/\* Create an IW Host Device \*/

h\_dev\_iwhost = WP\_DeviceCreate(h\_port\_iwhost, 0, WP\_DEVICE\_IW\_HOST, NULL);

App\_TerminateOnError(h\_dev\_iwhost, "WP\_DeviceCreate() IW Host");

/\* Create Buffer pools \*/

h\_pool\_buffer\_iw = WP\_PoolCreate(APP\_MASTER\_WPID, WP\_pool\_iwbuffer,

&pool\_buffer\_data\_iw\_cfg);

App\_TerminateOnError(h\_pool\_buffer\_iw, "WP\_PoolCreate iwbuffer iw");

/\* Pool for host termination buffers \*/

h\_pool\_256 = WP\_PoolCreate(WP\_WINPATH(0), WP\_pool\_buffer,

&pool\_buffer\_data\_256\_cfg);

App\_TerminateOnError (h\_pool\_256, "PoolCreate() pool\_256");

h\_pool\_ring\_host = WP\_PoolCreate(WP\_WINPATH(0), WP\_pool\_ring,

&pool\_ring\_data\_cfg);

App\_TerminateOnError(h\_pool\_ring\_host, "PoolCreate() pool\_ring\_host");

/\* Create an interworking queue node \*/

qnode\_iwq\_cfg.adjunct\_pool = h\_pool\_buffer\_iw;

h\_qnode\_iwq = WP\_QNodeCreate(APP\_MASTER\_WPID,WP\_QNODE\_IWQ | WP\_QNODE\_OPT\_DEDICATED\_RX\_HWQ | WP\_QNODE\_OPT\_FMU,

&qnode\_iwq\_cfg);

App\_TerminateOnError(h\_qnode\_iwq, "WP\_QNodeCreate() h\_qnode\_iwq");

/\* Create an interworking queue node \*/

qnode\_iwq\_cfg.adjunct\_pool = h\_pool\_buffer\_iw;

h\_qnode\_iwq\_tdm = WP\_QNodeCreate(APP\_MASTER\_WPID,WP\_QNODE\_IWQ, &qnode\_iwq\_cfg);

App\_TerminateOnError(h\_qnode\_iwq\_tdm, "WP\_QNodeCreate() h\_qnode\_iwq\_tdm");

/\* Host termination qnode \*/

qnode\_hostq\_cfg.pool\_buffer = h\_pool\_256;

qnode\_hostq\_cfg.pool\_ring = h\_pool\_ring\_host;

h\_qnode\_host = WP\_QNodeCreate(WP\_WINPATH(0), WP\_QNODE\_HOSTQ, &qnode\_hostq\_cfg);

App\_TerminateOnError(h\_qnode\_host, "QnodeCreate() h\_qnode\_mp");

status = WP\_ControlRegister(WP\_EVENT\_RX\_INDICATE, App\_EventRxIndicate);

App\_TerminateOnError(status, "WP\_ControlRegister()");

status = WP\_ControlRegister(WP\_EVENT\_TX\_INDICATE, App\_EventTxIndicate);

App\_TerminateOnError(status, "WP\_ControlRegister()");

status = WP\_ControlRegister(WP\_EVENT\_STATUS\_INDICATE, App\_EventErrorIndicate);

App\_TerminateOnError(status, "WP\_ControlRegister()");

status = WP\_SysSchedulerCreate(WP\_WINPATH(0), cw\_config);

App\_TerminateOnError(status, "WP\_SysSchedulerCreate()");

status = WP\_SysCommit(); /\* Commit system resources \*/

App\_TerminateOnError(status,"WP\_SysCommit()");

/\* Create Enet channels \*/

h\_enet1\_rx = WP\_ChannelCreate(tag\_enet1\_rx, h\_device\_enet1,

h\_qnode\_iwq,WP\_CH\_RX, WP\_ENET,

&ch\_enet\_cfg);

App\_TerminateOnError(h\_enet1\_rx, "WP\_ChannelCreate() Enet Rx");

/\* NOTE this channel is in NON-IW mode, used to send out a packet from the host out of ENET1 \*/

ch\_enet\_cfg.iwmode = WP\_PKTCH\_IWM\_DISABLE;

h\_enet1\_host\_tx = WP\_ChannelCreate(tag\_enet1\_tx++, h\_device\_enet1,

h\_qnode\_iwq,WP\_CH\_TX, WP\_ENET,

&ch\_enet\_cfg);

App\_TerminateOnError(h\_enet1\_host\_tx, "WP\_ChannelCreate() Enet Tx");

tx\_binding\_cfg.dci\_mode = WP\_IW\_DYN\_CH\_INSERT\_ENABLE;

status = WP\_IwTxBindingCreate(h\_enet1\_host\_tx, WP\_IW\_TX\_BINDING, &tx\_binding\_cfg);

App\_TerminateOnError(h\_enet1\_host\_tx, "WP\_IwTxBindingCreate()");

/\* Create Enet 2 channels \*/

ch\_enet\_cfg.iwmode = WP\_PKTCH\_IWM\_DISABLE;

h\_enet2\_rx = WP\_ChannelCreate(tag\_enet2\_rx, h\_device\_enet2,

h\_qnode\_iwq,WP\_CH\_RX, WP\_ENET,

&ch\_enet\_cfg);

App\_TerminateOnError(h\_enet2\_rx, "WP\_ChannelCreate() Enet Rx");

ch\_enet\_cfg.iwmode = WP\_PKTCH\_IWM\_ENABLE;

h\_enet2\_tx = WP\_ChannelCreate(tag\_enet2\_tx, h\_device\_enet2,

h\_qnode\_iwq,WP\_CH\_TX, WP\_ENET,

&ch\_enet\_cfg);

App\_TerminateOnError(h\_enet2\_tx, "WP\_ChannelCreate() Enet Tx");

tx\_binding\_cfg.dci\_mode = WP\_IW\_DYN\_CH\_INSERT\_ENABLE;

status = WP\_IwTxBindingCreate(h\_enet2\_tx, WP\_IW\_TX\_BINDING, &tx\_binding\_cfg);

App\_TerminateOnError(h\_enet2\_tx, "WP\_IwTxBindingCreate()");

tag1 = tag\_tdmrx;

tag2 = tag\_tdmtx;

/\* Create TDM RX and TX Channels \*/

for(ii = 0;ii < APP\_MAX\_TDM\_PORTS;ii++)

{

ch\_hdlc\_cfg.intmode = WP\_PKTCH\_INT\_ENABLE;

h\_tdm\_rx[ii] = WP\_ChannelCreate(tag1++, h\_dev\_tdm[ii], h\_qnode\_host,

WP\_CH\_RX, WP\_HDLC, &ch\_hdlc\_cfg);

App\_TerminateOnError(h\_tdm\_rx[ii], "WP\_ChannelCreate() h\_tdm\_rx");

ch\_hdlc\_cfg.intmode = WP\_PKTCH\_INT\_DISABLE;

h\_tdm\_tx[ii] = WP\_ChannelCreate(tag2++, h\_dev\_tdm[ii], h\_qnode\_iwq\_tdm,

WP\_CH\_TX, WP\_HDLC, &ch\_hdlc\_cfg);

App\_TerminateOnError(h\_tdm\_tx[ii], "WP\_ChannelCreate() h\_tdm\_tx1");

status = WP\_IwTxBindingCreate(h\_tdm\_tx[ii], WP\_IW\_TX\_BINDING, &tx\_binding\_cfg);

App\_TerminateOnError(status, "WP\_IwTxBindingCreate()");

}

WT\_Identify(); /\* Identify the testcase \*/

/\* Crate IW HOST termination channels one for each bridge \*/

tx\_binding\_cfg.dci\_mode = WP\_IW\_DYN\_CH\_INSERT\_DISABLE;

ch\_iw\_rx\_cfg.tx\_binding\_config = &tx\_binding\_cfg;

h\_bridge1\_iwhost = WP\_ChannelCreate(90,

h\_dev\_iwhost,

h\_qnode\_iwq,

WP\_CH\_RX,

WP\_IW\_HOST,

&ch\_iw\_rx\_cfg);

App\_TerminateOnError(h\_bridge1\_iwhost, "WP\_ChannelCreate() Host atm");

h\_bridge2\_iwhost = WP\_ChannelCreate(91,

h\_dev\_iwhost,

h\_qnode\_iwq,

WP\_CH\_RX,

WP\_IW\_HOST,

&ch\_iw\_rx\_cfg);

App\_TerminateOnError(h\_bridge2\_iwhost, "WP\_ChannelCreate() Host atm");

App\_CreateEnetHdlcBridge(); /\* Sets up ENET-->HDLC interworking \*/

App\_CreateHdlcEnetBridge(); /\* Sets up HDLC-->ENET interworking \*/

status = WP\_SysPsuEnable(WP\_WINPATH(0));

App\_TerminateOnError(status, "WP\_SysPsuEnable()");

/\* Enable all ports and devices \*/

status = WP\_PortEnable(h\_port\_enet1, WP\_DIRECTION\_DUPLEX);

App\_TerminateOnError(status, "WP\_PortEnable() Enet1");

status = WP\_DeviceEnable(h\_device\_enet1, WP\_DIRECTION\_DUPLEX);

App\_TerminateOnError(status, "WP\_DeviceEnable() Enet1");

status = WP\_PortEnable(h\_port\_enet2, WP\_DIRECTION\_DUPLEX);

App\_TerminateOnError(status, "WP\_PortEnable() Enet2");

status = WP\_DeviceEnable(h\_device\_enet2, WP\_DIRECTION\_DUPLEX);

App\_TerminateOnError(status, "WP\_DeviceEnable() Enet2");

for(ii = 0;ii < APP\_MAX\_TDM\_PORTS;ii++)

{

status = WP\_PortEnable(h\_port\_tdm[ii], WP\_DIRECTION\_DUPLEX);

App\_TerminateOnError(status,"WP\_PortEnable() TDM ");

status = WP\_DeviceEnable(h\_dev\_tdm[ii], WP\_DIRECTION\_DUPLEX);

App\_TerminateOnError(status,"WP\_DeviceEnable()");

}

status = WP\_ChannelEnable(h\_enet1\_rx);

App\_TerminateOnError(status, "WP\_ChannelEnable ENET1 RX");

status = WP\_ChannelEnable(h\_enet1\_host\_tx);

App\_TerminateOnError(status, "WP\_ChannelEnable ENET1 RX");

status = WP\_ChannelEnable(h\_enet2\_rx);

App\_TerminateOnError(status, "WP\_ChannelEnable ENET2 RX");

status = WP\_ChannelEnable(h\_enet2\_tx);

App\_TerminateOnError(status, "WP\_ChannelEnable ENET2 RX");

for(ii = 0;ii < APP\_MAX\_TDM\_PORTS;ii++)

{

status = WP\_ChannelEnable(h\_tdm\_tx[ii]);

App\_TerminateOnError(status, "WP\_ChannelEnable TDM TX");

}

while(1)

{

printf("a. Stats\n");

printf("d. Debug\n");

printf("p. Send Packet\n");

printf("s. Simulate Interrupts\n");

printf("x. Exit\n");

comm = getchar();

switch(comm)

{

case 'a':

App\_ShowStats();

break;

case 'd':

App\_Debug(debug\_on);

break;

case 'p':

App\_DataSend(h\_enet1\_host\_tx, h\_pool\_buffer\_iw);

break;

case 's':

WPI\_SimulateInterrupts();

break;

case 'x':

exit(0);

}

WPI\_SimulateInterrupts();

while((task = next\_task(irq\_task\_list, &a\_task)) != NULL)

app\_perform\_action(task);

}

return 0;

}

void App\_CreateHdlcEnetBridge(void)

{

WP\_U32 ii=0,jj=0,kk = 0;

WP\_status status;

WP\_U8 dummy\_prefix\_header[APP\_MAX\_TDM\_PORTS][14];

WP\_dfc\_flow\_info dfc\_flow\_info;

WP\_flow\_class\_rule flow\_class\_rule;

WP\_CHAR rule\_string[128];

WP\_classification\_filter classification\_filter;

WP\_flow\_class\_forwarding\_action forwarding\_action;

WP\_class\_filter\_user\_fields\_config filter\_user\_fields;

WP\_iw\_agg\_bridging iw\_agg\_bridging\_cfg;

/\* Dummy mac addresses used to send into Bridge2 \*/

for(ii = 0;ii < APP\_MAX\_TDM\_PORTS;ii++)

{

for(jj = 0;jj < 12;jj++)

{

dummy\_prefix\_header[ii][jj] = jj;

}

dummy\_prefix\_header[ii][12] = 0x08;

dummy\_prefix\_header[ii][13] = 0x00;

}

/\* PPP Switching System \*/

h\_iw\_sys\_pppsw = WP\_IwSystemCreate(WP\_WINPATH(0), WP\_IW\_PPP\_SWITCHING\_MODE,

&iw\_sys\_pppsw\_cfg);

App\_TerminateOnError(h\_iw\_sys\_pppsw, "WP\_IwSystemCreate() pppsw");

/\* Bridge2\*/

iw\_sys\_bridge.tag = tag\_iw\_sys\_bridge2;

h\_iw\_sys\_bridge2 = WP\_IwSystemCreate(APP\_MASTER\_WPID,

WP\_IW\_VLAN\_AWARE\_BRIDGING\_MODE,

&iw\_sys\_bridge);

App\_TerminateOnError(h\_iw\_sys\_bridge2," WP\_IwSystemCreate() Bridge2");

/\* Create 3 USER PROGRAMMABLE FILTERS \*/

/\* 1. FOR CISCO HDLC TRAFFIC CONTROL AND DATA \*/

classification\_filter.fields\_mask = WP\_FIELD\_IDS\_ARRAY;

classification\_filter.ip\_address\_config.ip\_sa\_prefix\_size = 32;

classification\_filter.ip\_address\_config.ip\_da\_prefix\_size = 32;

classification\_filter.filter\_type = WP\_CLASS\_FILTER\_EMC;

classification\_filter.field\_ids\_array[0] = WP\_FIELD\_ID\_IN\_PORT;

classification\_filter.field\_ids\_array[1] = WP\_FIELD\_ID\_USER\_PROGRAMMABLE;

classification\_filter.field\_ids\_array[2] = WP\_FIELD\_ID\_LAST;

classification\_filter.user\_fields\_config.num\_of\_user\_fields = 1;

classification\_filter.user\_fields\_config.ref\_point = WP\_CLASS\_REF\_POINT\_ETHER\_TYPE;

classification\_filter.user\_fields\_config.user\_fields[0].field\_offset = 0;

classification\_filter.user\_fields\_config.user\_fields[0].field\_size = 4;

classification\_filter.no\_match\_action = WP\_CLASS\_FILTER\_NO\_MATCH\_CONTINUE;

h\_bridge2\_filter[0] = WP\_IwClassFilterAdd(h\_iw\_sys\_bridge2, &classification\_filter);

App\_TerminateOnError(h\_bridge2\_filter[0]," WP\_IwClassFilterAdd() Bridge2");

/\* 2. FOR REGULAR PPP CONTROL TRAFFIC \*/

classification\_filter.user\_fields\_config.user\_fields[0].field\_size = 2;

classification\_filter.no\_match\_action = WP\_CLASS\_FILTER\_NO\_MATCH\_CONTINUE;

h\_bridge2\_filter[1] = WP\_IwClassFilterAdd(h\_iw\_sys\_bridge2, &classification\_filter);

App\_TerminateOnError(h\_bridge2\_filter[1]," WP\_IwClassFilterAdd() Bridge2");

/\* 3. FOR REGULAR PPP IP TRAFFIC \*/

classification\_filter.user\_fields\_config.user\_fields[0].field\_size = 1;

classification\_filter.no\_match\_action = WP\_CLASS\_FILTER\_NO\_MATCH\_DENY;

h\_bridge2\_filter[2] = WP\_IwClassFilterAdd(h\_iw\_sys\_bridge2, &classification\_filter);

App\_TerminateOnError(h\_bridge2\_filter[2]," WP\_IwClassFilterAdd() Bridge2");

memcpy(&iw\_agg\_bridging\_cfg, &default\_iw\_agg\_bridging\_cfg, sizeof(WP\_iw\_agg\_bridging));

iw\_agg\_bridging\_cfg.intmode = WP\_IW\_INT\_ENABLE;

/\* Create Default Aggregation for Bridge \*/

iw\_agg\_bridging\_cfg.tag = tag\_agg\_default\_bridge2;

iw\_agg\_bridging\_cfg.txfunc = h\_bridge2\_iwhost;

iw\_agg\_bridging\_cfg.input\_bport = 0;

iw\_agg\_bridging\_cfg.intmode = WP\_IW\_INT\_ENABLE;

h\_bridge2\_default\_agg = WP\_IwFlowAggregationCreate(

APP\_MASTER\_WPID,

WP\_IW\_VLAN\_AWARE\_BRIDGING\_MODE,

&iw\_agg\_bridging\_cfg);

App\_TerminateOnError(h\_bridge2\_default\_agg," WP\_IwFlowAggregationCreate() Bridge2 Default");

iw\_agg\_bridging\_cfg.intmode = WP\_IW\_INT\_DISABLE;

for(ii = 0;ii < APP\_MAX\_TDM\_PORTS;ii++)

{

WP\_rx\_binding\_ppp\_switching rx\_binding\_pppsw;

WP\_feature\_iw\_pppsw\_rx\_lcp\_forwarding pppsw\_lcp;

/\* Create a bridge port for each TDM port \*/

bridge\_port\_cfg.tag = tag\_bridge2\_bport\_tdm[ii];

bridge\_port\_cfg.flow\_agg = h\_bridge2\_default\_agg;

bridge\_port\_cfg.input\_filters\_mask = 0xe0000000;

bridge\_port\_cfg.classification\_flag = WP\_IW\_IWPORT\_CLASSIFICATION\_ENABLED;

h\_bridge2\_bport\_tdm[ii] = WP\_IwBridgingPortCreate(h\_iw\_sys\_bridge2,

&bridge\_port\_cfg);

App\_TerminateOnError(h\_bridge2\_bport\_tdm[ii],"WP\_IwBridgingPortCreate() Bridge2 TDM");

/\* Create a PPP Switching flow agg for each TDM port \*/

iw\_agg\_ppp\_switching.tag = tag\_agg\_pppsw[ii];

iw\_agg\_ppp\_switching.txfunc = h\_iw\_sys\_bridge2;

iw\_agg\_ppp\_switching.iw\_port = h\_bridge2\_bport\_tdm[ii];

iw\_agg\_ppp\_switching.prefix\_length = 14;

iw\_agg\_ppp\_switching.l2\_header\_insert\_mode = WP\_IW\_L2H\_INSERT\_ENABLE;

memcpy(&iw\_agg\_ppp\_switching.prefix\_header, &dummy\_prefix\_header[ii],14);

h\_flow\_agg\_pppsw\_link[ii] = WP\_IwFlowAggregationCreate(

WP\_WINPATH(0),

WP\_IW\_PPP\_SWITCHING\_MODE,

&iw\_agg\_ppp\_switching);

App\_TerminateOnError(h\_flow\_agg\_pppsw\_link[ii]," WP\_IwFlowAggregationCreate() PPPSW");

/\* Bind the TDM RX channel to the PPP Switching System using this flow agg \*/

rx\_binding\_pppsw.aggregation = h\_flow\_agg\_pppsw\_link[ii];

rx\_binding\_pppsw.mru = 1536;

rx\_binding\_pppsw.encap\_mode = WP\_IW\_VCMUX;

rx\_binding\_pppsw.acfc\_mode = 1;

rx\_binding\_pppsw.pfc\_mode = 1;

rx\_binding\_pppsw.pppmux\_ncp\_filter = WP\_IW\_PPPMUX\_NCP\_FILTER\_SWITCH;

rx\_binding\_pppsw.forward\_ppp\_to\_bundle\_mode = WP\_IW\_PPPSW\_FORWARD\_PPP\_TO\_LINK;

status = WP\_IwRxBindingCreate(h\_tdm\_rx[ii],

h\_iw\_sys\_pppsw,

h\_qnode\_iwq,

&rx\_binding\_pppsw);

App\_TerminateOnError(status," WP\_IwRxBindingCreate() PPPSW");

/\* ENABLE LCP forwarding on the TDM RX channel, so that LCP packets come into the PPP Switching system instead of being host terminated \*/

pppsw\_lcp.iw\_system = h\_iw\_sys\_pppsw;

pppsw\_lcp.aggregation = h\_flow\_agg\_pppsw\_link[ii];

status = WP\_FeatureInit(h\_tdm\_rx[ii], WP\_FEATURE\_IW\_PPPSW\_RX\_LCP\_FORWARDING\_MODE, &pppsw\_lcp);

App\_TerminateOnError(status , "WP\_FeatureInit LCP");

}

/\* Create a bridge port to represent ENET2 \*/

bridge\_port\_cfg.tag = tag\_bport\_enet2;

bridge\_port\_cfg.flow\_agg = h\_bridge2\_default\_agg;

bridge\_port\_cfg.input\_filters\_mask = 0;

bridge\_port\_cfg.classification\_flag = WP\_IW\_IWPORT\_CLASSIFICATION\_DISABLED;

h\_bridge2\_bport\_enet2 = WP\_IwBridgingPortCreate(h\_iw\_sys\_bridge2,

&bridge\_port\_cfg);

App\_TerminateOnError(h\_bridge2\_bport\_enet2,"WP\_IwBridgingPortCreate() ENET2");

/\* LCP flow agg - PPP LCP packets (0xC021) go here \*/

iw\_agg\_directmap\_cfg.tag = tag\_agg\_lcp;

iw\_agg\_directmap\_cfg.txfunc = h\_bridge2\_iwhost;

iw\_agg\_directmap\_cfg.iw\_port = 0;

iw\_agg\_directmap\_cfg.extraction\_length = 14;

iw\_agg\_directmap\_cfg.prefix\_length = 0;

iw\_agg\_directmap\_cfg.l2\_header\_insert\_mode = WP\_IW\_L2H\_INSERT\_DISABLE;

iw\_agg\_directmap\_cfg.intmode = WP\_IW\_INT\_ENABLE;

h\_flow\_agg\_lcp = WP\_IwFlowAggregationCreate(WP\_WINPATH(0), WP\_IW\_DIRECTMAP\_MODE,

&iw\_agg\_directmap\_cfg);

App\_TerminateOnError(h\_flow\_agg\_lcp," WP\_IwFlowAggregationCreate() LCP");

/\* Cisco HDLC Control Packets go here to this flow agg (0x0F008035) \*/

iw\_agg\_directmap\_cfg.tag = tag\_agg\_cisco\_control;

iw\_agg\_directmap\_cfg.txfunc = h\_bridge2\_iwhost;

iw\_agg\_directmap\_cfg.iw\_port = 0;

iw\_agg\_directmap\_cfg.extraction\_length = 14;

iw\_agg\_directmap\_cfg.prefix\_length = 0;

iw\_agg\_directmap\_cfg.l2\_header\_insert\_mode = WP\_IW\_L2H\_INSERT\_DISABLE;

iw\_agg\_directmap\_cfg.intmode = WP\_IW\_INT\_ENABLE;

h\_flow\_agg\_cisco\_control = WP\_IwFlowAggregationCreate(WP\_WINPATH(0), WP\_IW\_DIRECTMAP\_MODE,

&iw\_agg\_directmap\_cfg);

App\_TerminateOnError(h\_flow\_agg\_cisco\_control," WP\_IwFlowAggregationCreate() CC");

/\* Data packets go here - both PPP IP (0x21) and CISCO HDLC IP (0x0F000800) \*/

iw\_agg\_directmap\_cfg.tag = tag\_agg\_cisco\_control;

iw\_agg\_bridging\_cfg.tag = tag\_agg\_enet2;

iw\_agg\_bridging\_cfg.txfunc = h\_enet2\_tx;

iw\_agg\_bridging\_cfg.iw\_port = h\_bridge2\_bport\_enet2;

h\_flow\_agg\_enet2 = WP\_IwFlowAggregationCreate(

APP\_MASTER\_WPID,

WP\_IW\_VLAN\_AWARE\_BRIDGING\_MODE,

&iw\_agg\_bridging\_cfg);

App\_TerminateOnError(h\_flow\_agg\_enet2," WP\_IwFlowAggregationCreate() ENET2 ");

/\* Add DFC flows \*/

/\* NOTE for this test we are using TDM#1 to send out and receive the packet, so we use the bridge port for TDM 1 \*/

dfc\_flow\_info.input\_port = h\_bridge2\_bport\_tdm[1]; /\* Note that Input\_port is one of the fields in the filter \*/

dfc\_flow\_info.output\_port = 0;

/\* While adding these flows, make sure to use the correct filter for each flow \*/

/\* CISCO HDLC Control packets flow \*/

dfc\_flow\_info.filter\_handle = h\_bridge2\_filter[0];

strcpy(rule\_string, "n;n;n;n;n;n;#;0x0f008035");

forwarding\_action.flow\_aggregation = h\_flow\_agg\_cisco\_control;

flow\_class\_rule.classifier\_string = rule\_string;

flow\_class\_rule.dfc\_info = &dfc\_flow\_info;

flow\_class\_rule.match\_action = WP\_FLOW\_CLASS\_MODE\_FLOW\_AGG;

flow\_class\_rule.action\_info.forwarding\_action = forwarding\_action;

flow\_class\_rule.wred\_entry = 0;

h\_bridge2\_dfcflow[0] = WP\_IwFlowAdd(h\_iw\_sys\_bridge2, WP\_FLOW\_CLASS\_RULE, &flow\_class\_rule);

App\_TerminateOnError(h\_bridge2\_dfcflow[0]," WP\_IwFlowAdd() DFC Bridge2");

/\* CISCO HDLC Data packets flow \*/ /\* Same filter as above using 4 byte user fields \*/

dfc\_flow\_info.filter\_handle = h\_bridge2\_filter[0];

strcpy(rule\_string, "n;n;n;n;n;n;#;0x0f000800");

forwarding\_action.flow\_aggregation = h\_flow\_agg\_enet2;

flow\_class\_rule.classifier\_string = rule\_string;

flow\_class\_rule.dfc\_info = &dfc\_flow\_info;

flow\_class\_rule.match\_action = WP\_FLOW\_CLASS\_MODE\_FLOW\_AGG;

flow\_class\_rule.action\_info.forwarding\_action = forwarding\_action;

flow\_class\_rule.wred\_entry = 0;

h\_bridge2\_dfcflow[1] = WP\_IwFlowAdd(h\_iw\_sys\_bridge2, WP\_FLOW\_CLASS\_RULE, &flow\_class\_rule);

App\_TerminateOnError(h\_bridge2\_dfcflow[1]," WP\_IwFlowAdd() DFC Bridge2");

/\* PPP LCP flow \*/ /\* Note the use of FILTER 1 \*/

dfc\_flow\_info.filter\_handle = h\_bridge2\_filter[1];

strcpy(rule\_string, "n;n;n;n;n;n;#;0xc021");

forwarding\_action.flow\_aggregation = h\_flow\_agg\_lcp;

flow\_class\_rule.classifier\_string = rule\_string;

flow\_class\_rule.dfc\_info = &dfc\_flow\_info;

flow\_class\_rule.match\_action = WP\_FLOW\_CLASS\_MODE\_FLOW\_AGG;

flow\_class\_rule.action\_info.forwarding\_action = forwarding\_action;

flow\_class\_rule.wred\_entry = 0;

h\_bridge2\_dfcflow[2] = WP\_IwFlowAdd(h\_iw\_sys\_bridge2, WP\_FLOW\_CLASS\_RULE, &flow\_class\_rule);

App\_TerminateOnError(h\_bridge2\_dfcflow[2]," WP\_IwFlowAdd() DFC Bridge2");

/\* PPP IP flow \*/ /\* Note the use of FILTER 2 \*/

dfc\_flow\_info.filter\_handle = h\_bridge2\_filter[2];

strcpy(rule\_string, "n;n;n;n;n;n;#;0x21");

forwarding\_action.flow\_aggregation = h\_flow\_agg\_enet2;

flow\_class\_rule.classifier\_string = rule\_string;

flow\_class\_rule.dfc\_info = &dfc\_flow\_info;

flow\_class\_rule.match\_action = WP\_FLOW\_CLASS\_MODE\_FLOW\_AGG;

flow\_class\_rule.action\_info.forwarding\_action = forwarding\_action;

flow\_class\_rule.wred\_entry = 0;

h\_bridge2\_dfcflow[3] = WP\_IwFlowAdd(h\_iw\_sys\_bridge2, WP\_FLOW\_CLASS\_RULE, &flow\_class\_rule);

App\_TerminateOnError(h\_bridge2\_dfcflow[3]," WP\_IwFlowAdd() DFC Bridge2");

/\* Be sure to build the system \*/

status = WP\_IwSystemBuild(h\_iw\_sys\_bridge2);

App\_TerminateOnError(status," WP\_IwSystemBuild() Bridge2");

}

void App\_CreateEnetHdlcBridge()

{

WP\_U32 ii=0,jj=0,kk = 0;

WP\_status status;

WP\_flow\_route\_ipv4 flow\_route\_cfg;

WP\_dfc\_flow\_info dfc\_flow\_info;

WP\_flow\_class\_rule flow\_class\_rule;

WP\_CHAR rule\_string[128];

WP\_classification\_filter classification\_filter;

WP\_flow\_class\_forwarding\_action forwarding\_action;

WP\_iw\_agg\_bridging iw\_agg\_bridging\_cfg;

/\* Bridge1 ENET --> TDM \*/

iw\_sys\_bridge.tag = tag\_iw\_sys\_bridge1;

h\_iw\_sys\_bridge1 = WP\_IwSystemCreate(APP\_MASTER\_WPID,

WP\_IW\_VLAN\_AWARE\_BRIDGING\_MODE,

&iw\_sys\_bridge);

App\_TerminateOnError(h\_iw\_sys\_bridge1," WP\_IwSystemCreate() Bridge1");

/\* DFC filter using 3 fields \*/

classification\_filter.fields\_mask = WP\_FIELD\_IDS\_ARRAY;

classification\_filter.no\_match\_action = WP\_CLASS\_FILTER\_NO\_MATCH\_DENY;

/\* Using 3 fields, include the input bridge port \*/

classification\_filter.field\_ids\_array[0] = WP\_FIELD\_ID\_VLAN\_ID;

classification\_filter.field\_ids\_array[1] = WP\_FIELD\_ID\_ETHER\_TYPE;

classification\_filter.field\_ids\_array[2] = WP\_FIELD\_ID\_IN\_PORT;

classification\_filter.field\_ids\_array[3] = WP\_FIELD\_ID\_LAST;

classification\_filter.ip\_address\_config.ip\_sa\_prefix\_size = 32;

classification\_filter.ip\_address\_config.ip\_da\_prefix\_size = 32;

classification\_filter.filter\_type = WP\_CLASS\_FILTER\_EMC;

h\_bridge1\_filter1 = WP\_IwClassFilterAdd(h\_iw\_sys\_bridge1, &classification\_filter);

App\_TerminateOnError(h\_bridge1\_filter1," WP\_IwClassFilterAdd() Bridge1");

memcpy(&iw\_agg\_bridging\_cfg, &default\_iw\_agg\_bridging\_cfg, sizeof(WP\_iw\_agg\_bridging));

/\* Create Default Aggregation Bridge1 \*/

iw\_agg\_bridging\_cfg.tag = tag\_agg\_default\_bridge1;

iw\_agg\_bridging\_cfg.txfunc = h\_bridge1\_iwhost;

iw\_agg\_bridging\_cfg.input\_bport = 0;

iw\_agg\_bridging\_cfg.intmode = WP\_IW\_INT\_ENABLE;

h\_bridge1\_default\_agg = WP\_IwFlowAggregationCreate(

APP\_MASTER\_WPID,

WP\_IW\_VLAN\_AWARE\_BRIDGING\_MODE,

&iw\_agg\_bridging\_cfg);

App\_TerminateOnError(h\_bridge1\_default\_agg," WP\_IwFlowAggregationCreate() Default Bridge");

iw\_agg\_bridging\_cfg.intmode = WP\_IW\_INT\_DISABLE;

bridge\_port\_cfg.tag = tag\_bport\_enet1;

bridge\_port\_cfg.flow\_agg = h\_bridge1\_default\_agg;

bridge\_port\_cfg.input\_filters\_mask = 0x80000000;

h\_bridge1\_bport\_enet1 = WP\_IwBridgingPortCreate(h\_iw\_sys\_bridge1,

&bridge\_port\_cfg);

App\_TerminateOnError(h\_bridge1\_bport\_enet1,"WP\_IwBridgingPortCreate()ENET1");

/\* Rx binding ENET 1 \*/

rx\_binding\_bridging\_cfg.input\_port = h\_bridge1\_bport\_enet1;

status = WP\_IwRxBindingCreate(h\_enet1\_rx,

h\_iw\_sys\_bridge1,

h\_qnode\_iwq,

&rx\_binding\_bridging\_cfg);

App\_TerminateOnError(status," WP\_IwRxBindingCreate() enet1 ");

/\* EACH TDM port is represented as bridge port in Bridge1 too so a packet going from ENET to TDM has the ENET bridge port as input bridge port and TDM

\* bridge port as the output bridge port \*/

for(ii = 0;ii < APP\_MAX\_TDM\_PORTS;ii++)

{

bridge\_port\_cfg.tag = tag\_bridge1\_bport\_tdm[ii];

bridge\_port\_cfg.flow\_agg = h\_bridge1\_default\_agg;

bridge\_port\_cfg.max\_mac\_addresses = APP\_MAX\_MAC\_ADDRESS;

bridge\_port\_cfg.input\_filters\_mask = 0;

bridge\_port\_cfg.classification\_flag = WP\_IW\_IWPORT\_CLASSIFICATION\_DISABLED;

h\_bridge1\_bport\_tdm[ii] = WP\_IwBridgingPortCreate(h\_iw\_sys\_bridge1,

&bridge\_port\_cfg);

App\_TerminateOnError(h\_bridge1\_bport\_tdm[ii],"WP\_IwBridgingPortCreate()TDM");

}

/\* Create flow aggregations that send the packet out on each TDM \*/

/\* NOTE THE 4 BYTE HDLC header \*/ /\* To test different scenarios you need to change the 4 byte header for now \*/

/\* ALSO NOTE the 18 bytes being extracted \*/

for(ii = 0;ii < APP\_MAX\_TDM\_PORTS;ii++)

{

WP\_U8 hdlc\_prefix\_header[4] = {0x0f,0x00,0x80,0x35};

iw\_agg\_bridging\_cfg.tag = tag\_agg\_tdm[ii];

iw\_agg\_bridging\_cfg.txfunc = h\_tdm\_tx[ii];

iw\_agg\_bridging\_cfg.iw\_port = h\_bridge1\_bport\_tdm[ii];

iw\_agg\_bridging\_cfg.extraction\_length = 18;

iw\_agg\_bridging\_cfg.prefix\_length = 4;

iw\_agg\_bridging\_cfg.l2\_header\_insert\_mode = WP\_IW\_L2H\_INSERT\_ENABLE;

memcpy(iw\_agg\_bridging\_cfg.prefix\_header, hdlc\_prefix\_header, 4);

h\_flow\_agg\_tdm[ii] = WP\_IwFlowAggregationCreate(

APP\_MASTER\_WPID,

WP\_IW\_VLAN\_AWARE\_BRIDGING\_MODE,

&iw\_agg\_bridging\_cfg);

App\_TerminateOnError(h\_flow\_agg\_tdm[ii]," WP\_IwFlowAggregationCreate() TDM ");

}

iw\_agg\_bridging\_cfg.prefix\_length = 0;

iw\_agg\_bridging\_cfg.l2\_header\_insert\_mode = WP\_IW\_L2H\_INSERT\_DISABLE;

iw\_agg\_bridging\_cfg.extraction\_length = 0;

/\* Add DFC flow \*/

dfc\_flow\_info.filter\_handle = h\_bridge1\_filter1;

dfc\_flow\_info.input\_port = h\_bridge1\_bport\_enet1; /\* Note that Input\_port is one of the fields in the filter \*/

dfc\_flow\_info.output\_port = 0;

forwarding\_action.flow\_aggregation = h\_flow\_agg\_tdm[1];

/\* Add VLAN 10 ETHER TYPE = 0x800\*/

/\* Note: Input Bridge port not passed in string but passed in above \*/

sprintf(rule\_string, "n;n;n;n;n;n;n;n;n;n;10;n;n;0x800;n;n;n;n;n;n;n;n;n;n;n;n;n;n;n;n;n;n");

flow\_class\_rule.classifier\_string = rule\_string;

flow\_class\_rule.dfc\_info = &dfc\_flow\_info;

flow\_class\_rule.match\_action = WP\_FLOW\_CLASS\_MODE\_FLOW\_AGG;

flow\_class\_rule.action\_info.forwarding\_action = forwarding\_action;

flow\_class\_rule.wred\_entry = 0;

h\_bridge1\_dfcflow1 = WP\_IwFlowAdd(h\_iw\_sys\_bridge1, WP\_FLOW\_CLASS\_RULE, &flow\_class\_rule);

App\_TerminateOnError(h\_bridge1\_dfcflow1," WP\_IwFlowAdd() DFC Bridge1");

status = WP\_IwSystemBuild(h\_iw\_sys\_bridge1);

App\_TerminateOnError(status," WP\_IwSystemBuild() Bridge1");

}

/\* Send a Ethernet packet on the channel handle passed in \*/

void App\_DataSend(WP\_handle h\_tx, WP\_handle h\_pool)

{

#define APP\_DATA\_LENGTH 80

WP\_data\_unit data\_unit = {0} ;

WP\_data\_segment bufs = {0};

WP\_status status;

WP\_U8 \* km\_data\_ptr;

WP\_U32 ii;

WP\_U8 dst\_mac[6] = {0x00,0x00,0x00,0x00,0x00,0x01};

WP\_U8 src\_mac[6] = {0x00,0x00,0x00,0x00,0x00,0x02};

WP\_U8 vlan\_1[4] = {0x81,0x00,0x00,0xa};

WP\_U8 e\_type[2] = {0x08,0x00};

WP\_U32 l2\_header\_len = sizeof(dst\_mac) + sizeof(src\_mac) + sizeof(vlan\_1) +

sizeof(e\_type);

WP\_U32 ip\_header\_len = 20;

WP\_U32 payload\_len;

WP\_U32 offset;

bufs.buffer\_class = 0;

bufs.displacement = 0;

bufs.user\_info = 0;

bufs.next = NULL;

bufs.pool\_handle = h\_pool;

bufs.data\_size = APP\_DATA\_LENGTH;

data\_unit.type = WP\_DATA\_ENET;

data\_unit.n\_active = 1;

data\_unit.data\_size = bufs.data\_size;

data\_unit.segment = &bufs;

data\_unit.n\_segments = 1;

km\_data\_ptr = WP\_PoolAlloc(h\_pool);

bufs.data = km\_data\_ptr;

if(bufs.data == NULL)

{

printf("Pool Alloc failure ! \n");

}

WP\_MEM\_BYTES\_FILL(km\_data\_ptr, APP\_DATA\_LENGTH, pool\_buffer\_data\_iw\_cfg.size);

offset = 0;

memcpy(km\_data\_ptr + offset,dst\_mac,sizeof(dst\_mac));

offset += sizeof(dst\_mac);

memcpy(km\_data\_ptr+offset,src\_mac,sizeof(src\_mac));

offset += sizeof(src\_mac);

memcpy(km\_data\_ptr+offset,vlan\_1,sizeof(vlan\_1));

offset += sizeof(vlan\_1);

memcpy(km\_data\_ptr+offset,e\_type,sizeof(e\_type));

offset += sizeof(e\_type);

payload\_len = APP\_DATA\_LENGTH - l2\_header\_len - ip\_header\_len;

/\* Outer IP header \*/

insert\_ip\_header(0x0a0a0a0a,0xc0a8fe11,payload\_len,4,offset,km\_data\_ptr);

offset += ip\_header\_len;

memset(km\_data\_ptr+offset,0xa,payload\_len);

printf("sending packet of length %d\n",APP\_DATA\_LENGTH);

for(ii = 0;ii < APP\_DATA\_LENGTH;ii++)

{

printf("%.2x",\*(km\_data\_ptr+ii));

if((ii != 0) && !(ii % 40))

printf("\n");

}

printf("\n");

/\* Use interworking qnode and interworking buffer pool for Gige \*/

status = WP\_HostSend(h\_tx, &data\_unit);

App\_TerminateOnError(status, "WP\_HostSend()");

WP\_Delay(500000);

WPI\_SimulateInterrupts();

}

/\* Event handling \*/

void add\_task(app\_task\_list \*task\_list, WP\_U32 event\_type, WP\_tag event\_tag,

void \*event\_param)

{

WP\_U16 tail = task\_list->tail;

WP\_U32 next = tail + 1;

if (next == task\_list->num\_elements)

next = 0;

if (next != task\_list->head) {

task\_list->task[tail].event\_tag = event\_tag;

task\_list->task[tail].event\_type = event\_type;

task\_list->task[tail].event\_param = event\_param;

task\_list->tail = next;

}

}

/\* Event handling \*/

app\_task \*

next\_task(app\_task\_list \*task\_list, app\_task \*result)

{

WP\_U32 head = task\_list->head;

WP\_U32 tail = task\_list->tail;

if (head == tail)

return NULL;

\*result = task\_list->task[head];

if (++head == task\_list->num\_elements)

head = 0;

task\_list->head = head;

return result;

}

/\* Event handling \*/

void app\_perform\_action(app\_task \*task)

{

WP\_U32 tag = task->event\_tag;

switch(task->event\_type)

{

case WP\_EVENT\_RX\_INDICATE:

{

printf("Rx event on tag %d \n",tag);

if(tag == tag\_enet1\_rx)

{

printf("receiving on enet1\n");

App\_DuReceive(h\_enet1\_rx, WP\_DATA\_ENET);

}

else if(tag == tag\_enet2\_rx)

{

printf("receiving on enet2\n");

App\_DuReceive(h\_enet2\_rx, WP\_DATA\_ENET);

}

else if(tag == tag\_agg\_default\_bridge1)

{

printf("receiving on default bridge1\n");

App\_DuReceive(h\_bridge1\_iwhost, WP\_DATA\_IW);

}

else if(tag == tag\_agg\_default\_bridge2)

{

printf("receiving on default bridge2\n");

App\_DuReceive(h\_bridge2\_iwhost, WP\_DATA\_IW);

}

else if(tag == tag\_agg\_lcp)

{

printf("receiving on PPP LCP aggregation\n");

App\_DuReceive(h\_bridge2\_iwhost, WP\_DATA\_IW);

}

else if(tag == tag\_agg\_cisco\_control)

{

printf("receiving on Cisco Control Packet Aggregation\n");

App\_DuReceive(h\_bridge2\_iwhost, WP\_DATA\_IW);

}

else if((tag >= tag\_tdmrx) && (tag <= tag\_tdmrx + APP\_MAX\_TDM\_PORTS))

{

printf("receiving on tdm %d\n",tag - tag\_tdmrx);

App\_DuReceive(h\_tdm\_rx[tag - tag\_tdmrx], WP\_DATA\_PPP\_HDLC);

}

else

{

printf("RX event on unknown tag %d\n",tag);

}

break;

}

case WP\_EVENT\_TX\_INDICATE:

{

printf("TX event on tag %d\n",tag);

break;

}

case WP\_EVENT\_STATUS\_INDICATE:

break;

default:

printf("Not processing unknown event\n");

}

}

/\* Receives a packet on a RX channel \*/

void App\_DuReceive(WP\_handle h\_rx, WP\_U32 data\_type)

{

WP\_data\_unit data\_unit;

WP\_data\_segment data\_segment, \*curr\_buff\_ptr;

WP\_U32 kk;

WP\_status status;

data\_unit.segment = &data\_segment;

data\_unit.n\_segments = 1;

data\_unit.type = data\_type;

data\_unit.n\_active = 0;

status = WP\_HostReceive(h\_rx, &data\_unit);

App\_TerminateOnError(status, "WP\_HostReceive");

if(data\_unit.n\_active > 0)

{

curr\_buff\_ptr = data\_unit.segment;

for(kk = 0;kk < curr\_buff\_ptr->data\_size;kk++)

{

printf("%2.2x", curr\_buff\_ptr->data[kk]);

}

printf("( %d bytes )\n", curr\_buff\_ptr->data\_size);

}

}

void App\_EventRxIndicate(WP\_tag tag, WP\_U32 data, WP\_U32 info)

{

add\_task(irq\_task\_list, WP\_EVENT\_RX\_INDICATE, tag, NULL);

}

void App\_EventTxIndicate(WP\_tag tag, WP\_U32 data, WP\_U32 info)

{

add\_task(irq\_task\_list, WP\_EVENT\_TX\_INDICATE, tag, NULL);

}

void App\_EventErrorIndicate(WP\_tag tag, WP\_U32 event, WP\_U32 data)

{

add\_task(irq\_task\_list, WP\_EVENT\_STATUS\_INDICATE, tag, NULL);

}

void App\_Quit(void)

{

WP\_DriverRelease();

#if \_WT\_MODULE\_TEST\_ENABLE\_

WT\_Reboot();

#endif

exit(0);

}

void App\_TerminateOnError(WP\_handle handle, WP\_CHAR \*s)

{

if (WPI\_SimulateInterrupts(), WP\_ERROR\_P(handle)) {

printf("Test Abort %s %s 0x%x\n", s, WP\_error\_name[WP\_ERROR(handle)], handle);

/\* Release WDDI \*/

WP\_DriverRelease();

#if \_WT\_MODULE\_TEST\_ENABLE\_

WT\_Reboot();

#endif

exit(1);

}

else

{

#if 1

if(handle == WP\_OK)

printf("Status returned from %s : WP\_OK\n",s);

else

printf("Handle returned from %s is %#8.8x\n", s,handle);

#endif

}

}

void App\_ShowStats(void)

{

WP\_status status;

WP\_stats\_enet s\_hs\_enet = {0};

WP\_iw\_global\_stats s\_iw\_global = {0};

WP\_iw\_flow\_stats flow\_stats = {0};

WP\_bport\_stats bport\_stats = {0};

WP\_route\_port\_stats rport\_stats = {0};

WP\_stats\_tdi\_hdlc tdi\_hdlc\_stats = {0};

WP\_U32 ii;

memset(&s\_hs\_enet,0,sizeof(s\_hs\_enet));

printf("-------------------------------\n");

printf(" ENET1 statisics \n");

printf("-------------------------------\n");

status = WP\_DeviceStatistics(h\_device\_enet1, &s\_hs\_enet);

App\_TerminateOnError(status, "WP\_DeviceStatistics");

printf(" rx\_bytes : %u\n", (WP\_U32)s\_hs\_enet.rx\_bytes);

printf(" rx\_packets : %u\n", (WP\_U32)s\_hs\_enet.rx\_packets);

printf(" rx\_err\_fcs : %u\n", (WP\_U32)s\_hs\_enet.rx\_err\_fcs);

printf(" rx\_multicast : %u\n", (WP\_U32)s\_hs\_enet.rx\_multicast);

printf(" rx\_broadcast : %u\n", (WP\_U32)s\_hs\_enet.rx\_broadcast);

printf(" rx\_dropped : %u\n\n", (WP\_U32)s\_hs\_enet.rx\_dropped);

printf(" tx\_bytes : %u\n", (WP\_U32)s\_hs\_enet.tx\_bytes);

printf(" tx\_packets : %u\n", (WP\_U32)s\_hs\_enet.tx\_packets);

printf(" tx\_err\_fcs : %u\n", (WP\_U32)s\_hs\_enet.tx\_err\_fcs);

printf(" tx\_multicast : %u\n", (WP\_U32)s\_hs\_enet.tx\_multicast);

printf(" tx\_broadcast : %u\n", (WP\_U32)s\_hs\_enet.tx\_broadcast);

printf(" tx\_dropped : %u\n\n", (WP\_U32)s\_hs\_enet.tx\_dropped);

printf(" rx\_err\_overrun : %u\n", (WP\_U32)s\_hs\_enet.rx\_err\_overrun);

printf(" rx\_err\_mru : %u\n", (WP\_U32)s\_hs\_enet.rx\_err\_mru);

printf(" rx\_iw\_frames : %u\n", (WP\_U32)s\_hs\_enet.rx\_iw\_frames);

printf(" rx\_host\_frames : %u\n", (WP\_U32)s\_hs\_enet.rx\_host\_frames);

printf(" tx\_frames : %u\n", (WP\_U32)s\_hs\_enet.tx\_frames);

memset(&s\_hs\_enet,0,sizeof(s\_hs\_enet));

printf("-------------------------------\n");

printf(" ENET2 statisics \n");

printf("-------------------------------\n");

status = WP\_DeviceStatistics(h\_device\_enet2, &s\_hs\_enet);

App\_TerminateOnError(status, "WP\_DeviceStatistics");

printf(" rx\_bytes : %u\n", (WP\_U32)s\_hs\_enet.rx\_bytes);

printf(" rx\_packets : %u\n", (WP\_U32)s\_hs\_enet.rx\_packets);

printf(" rx\_err\_fcs : %u\n", (WP\_U32)s\_hs\_enet.rx\_err\_fcs);

printf(" rx\_multicast : %u\n", (WP\_U32)s\_hs\_enet.rx\_multicast);

printf(" rx\_broadcast : %u\n", (WP\_U32)s\_hs\_enet.rx\_broadcast);

printf(" rx\_dropped : %u\n\n", (WP\_U32)s\_hs\_enet.rx\_dropped);

printf(" tx\_bytes : %u\n", (WP\_U32)s\_hs\_enet.tx\_bytes);

printf(" tx\_packets : %u\n", (WP\_U32)s\_hs\_enet.tx\_packets);

printf(" tx\_err\_fcs : %u\n", (WP\_U32)s\_hs\_enet.tx\_err\_fcs);

printf(" tx\_multicast : %u\n", (WP\_U32)s\_hs\_enet.tx\_multicast);

printf(" tx\_broadcast : %u\n", (WP\_U32)s\_hs\_enet.tx\_broadcast);

printf(" tx\_dropped : %u\n\n", (WP\_U32)s\_hs\_enet.tx\_dropped);

printf(" rx\_err\_overrun : %u\n", (WP\_U32)s\_hs\_enet.rx\_err\_overrun);

printf(" rx\_err\_mru : %u\n", (WP\_U32)s\_hs\_enet.rx\_err\_mru);

printf(" rx\_iw\_frames : %u\n", (WP\_U32)s\_hs\_enet.rx\_iw\_frames);

printf(" rx\_host\_frames : %u\n", (WP\_U32)s\_hs\_enet.rx\_host\_frames);

printf(" tx\_frames : %u\n", (WP\_U32)s\_hs\_enet.tx\_frames);

printf("-------------------------------\n");

printf("Bridge Port Statistics ENET1 \n");

printf("-------------------------------\n");

memset(&bport\_stats, 0, sizeof(bport\_stats));

status = WP\_IwPortStatistics(h\_bridge1\_bport\_enet1, &bport\_stats);

App\_TerminateOnError(status, "WP\_BportStatistics");

printf("rx\_valid\_packets = %d\n",(WP\_U32)bport\_stats.rx\_valid\_packets);

printf("discard\_vlan\_acceptable\_filter = %d\n",(WP\_U32)bport\_stats.discard\_vlan\_acceptable\_filter);

printf("discard\_ingress\_filter = %d\n",(WP\_U32)bport\_stats.discard\_ingress\_filter);

printf("discard\_bridge\_classifier = %d\n",(WP\_U32)bport\_stats.discard\_bridge\_classifier);

printf("discard\_unk\_macsa = %d\n",(WP\_U32)bport\_stats.discard\_unk\_macsa);

printf("deny\_mac\_sa = %d\n",(WP\_U32)bport\_stats.deny\_mac\_sa);

printf("deny\_mac\_da = %d\n",(WP\_U32)bport\_stats.deny\_mac\_da);

printf("rx\_bc\_valid\_packets = %d\n",(WP\_U32)bport\_stats.rx\_bc\_valid\_packets);

printf("rx\_mc\_valid\_packets = %d\n",(WP\_U32)bport\_stats.rx\_mc\_valid\_packets);

printf("forwarded\_uc\_packets = %d\n",(WP\_U32)bport\_stats.forwarded\_uc\_packets);

printf("forwarded\_bc\_packets = %d\n",(WP\_U32)bport\_stats.forwarded\_bc\_packets);

printf("forwarded\_mc\_packets = %d\n",(WP\_U32)bport\_stats.forwarded\_mc\_packets);

printf("protocol\_error = %d\n",(WP\_U32)bport\_stats.protocol\_error);

for(ii = 0;ii < APP\_MAX\_TDM\_PORTS;ii++)

{

printf("-------------------------------\n");

printf("Bridge Port Statistics TDM %d\n",ii);

printf("-------------------------------\n");

memset(&bport\_stats, 0, sizeof(bport\_stats));

status = WP\_IwPortStatistics(h\_bridge1\_bport\_tdm[ii], &bport\_stats);

App\_TerminateOnError(status, "BportStatistics");

printf("rx\_valid\_packets = %d\n",(WP\_U32)bport\_stats.rx\_valid\_packets);

printf("discard\_vlan\_acceptable\_filter = %d\n",(WP\_U32)bport\_stats.discard\_vlan\_acceptable\_filter);

printf("discard\_ingress\_filter = %d\n",(WP\_U32)bport\_stats.discard\_ingress\_filter);

printf("discard\_bridge\_classifier = %d\n",(WP\_U32)bport\_stats.discard\_bridge\_classifier);

printf("discard\_unk\_macsa = %d\n",(WP\_U32)bport\_stats.discard\_unk\_macsa);

printf("deny\_mac\_sa = %d\n",(WP\_U32)bport\_stats.deny\_mac\_sa);

printf("deny\_mac\_da = %d\n",(WP\_U32)bport\_stats.deny\_mac\_da);

printf("rx\_bc\_valid\_packets = %d\n",(WP\_U32)bport\_stats.rx\_bc\_valid\_packets);

printf("rx\_mc\_valid\_packets = %d\n",(WP\_U32)bport\_stats.rx\_mc\_valid\_packets);

printf("forwarded\_uc\_packets = %d\n",(WP\_U32)bport\_stats.forwarded\_uc\_packets);

printf("forwarded\_bc\_packets = %d\n",(WP\_U32)bport\_stats.forwarded\_bc\_packets);

printf("forwarded\_mc\_packets = %d\n",(WP\_U32)bport\_stats.forwarded\_mc\_packets);

printf("protocol\_error = %d\n",(WP\_U32)bport\_stats.protocol\_error);

}

for(ii = 0;ii < APP\_MAX\_TDM\_PORTS;ii++)

{

printf("-------------------------------\n");

printf(" Flow statistics TDM %d\n",ii);

printf("-------------------------------\n");

memset(&flow\_stats,0,sizeof(flow\_stats));

status = WP\_IwFlowStatistics(h\_flow\_agg\_tdm[ii],WP\_IW\_FLOW\_STAT,&flow\_stats);

App\_TerminateOnError(status, "WP\_IwFlowStatistics");

printf("forward\_packet = %d\n",(WP\_U32)flow\_stats.forward\_packet);

printf("fbp\_drop\_packets = %d\n",(WP\_U32)flow\_stats.fbp\_drop\_packets);

printf("mtu\_drop\_packets = %d\n",(WP\_U32)flow\_stats.mtu\_drop\_packets);

printf("ttl\_drop\_packets = %d\n",(WP\_U32)flow\_stats.ttl\_drop\_packets);

printf("tx\_queue\_drop\_packets = %d\n",(WP\_U32)flow\_stats.tx\_queue\_drop\_packets);

printf("mpls\_drop = %d\n",(WP\_U32)flow\_stats.mpls\_drop);

printf("denied\_packets = %d\n",(WP\_U32)flow\_stats.denied\_packets);

printf("group\_filtered\_packets = %d\n",(WP\_U32)flow\_stats.group\_filtered\_packets);

printf("forwarded\_bytes = %d\n",(WP\_U32)flow\_stats.forwarded\_bytes);

printf("gtp\_bad\_headers = %d\n",(WP\_U32)flow\_stats.gtp\_bad\_headers);

printf("policer\_non\_conforming\_packets; = %d\n",(WP\_U32)flow\_stats.policer\_non\_conforming\_packets);

}

for(ii = 0;ii < APP\_MAX\_TDM\_PORTS;ii++)

{

printf("-------------------------------\n");

printf(" Device statisics TDM %d \n",ii);

printf("-------------------------------\n");

memset(&tdi\_hdlc\_stats,0,sizeof(WP\_stats\_tdi\_hdlc));

status = WP\_DeviceStatistics(h\_dev\_tdm[ii],&tdi\_hdlc\_stats);

App\_TerminateOnError(status, "WP\_Devicestatistics");

printf("rx\_frames = %d\n", (WP\_U32)tdi\_hdlc\_stats.rx\_frames);

printf("tx\_frames = %d\n", (WP\_U32)tdi\_hdlc\_stats.tx\_frames);

printf("rx\_err\_crc = %d\n", (WP\_U32)tdi\_hdlc\_stats.rx\_err\_crc);

printf("rx\_err\_abort = %d\n", (WP\_U32)tdi\_hdlc\_stats.rx\_err\_abort);

printf("rx\_err\_addr\_mismatch = %d\n", (WP\_U32)tdi\_hdlc\_stats.rx\_err\_addr\_mismatch);

printf("rx\_err\_buffer\_overrun = %d\n", (WP\_U32)tdi\_hdlc\_stats.rx\_err\_buffer\_overrun);

printf("rx\_err\_overrun = %d\n", (WP\_U32)tdi\_hdlc\_stats.rx\_err\_overrun);

printf("rx\_err\_iw\_buffer\_underrun = %d\n", (WP\_U32)tdi\_hdlc\_stats.rx\_err\_iw\_buffer\_underrun);

printf("rx\_err\_iw\_mru = %d\n", (WP\_U32)tdi\_hdlc\_stats.rx\_err\_iw\_mru);

printf("tx\_err\_underrun = %d\n", (WP\_U32)tdi\_hdlc\_stats.tx\_err\_underrun);

printf("rx\_err\_maxsdu = %d\n", (WP\_U32)tdi\_hdlc\_stats.rx\_err\_maxsdu);

printf("rx\_iw\_frames = %d\n", (WP\_U32)tdi\_hdlc\_stats.rx\_iw\_frames);

printf("rx\_err\_non\_octet = %d\n", (WP\_U32)tdi\_hdlc\_stats.rx\_err\_non\_octet);

}

for(ii = 0;ii < APP\_MAX\_TDM\_PORTS;ii++)

{

printf("-------------------------------\n");

printf(" Flow statistics PPPSW %d\n",ii);

printf("-------------------------------\n");

memset(&flow\_stats,0,sizeof(flow\_stats));

status = WP\_IwFlowStatistics(h\_flow\_agg\_pppsw\_link[ii],WP\_IW\_FLOW\_STAT,&flow\_stats);

App\_TerminateOnError(status, "WP\_IwFlowStatistics");

printf("forward\_packet = %d\n",(WP\_U32)flow\_stats.forward\_packet);

printf("fbp\_drop\_packets = %d\n",(WP\_U32)flow\_stats.fbp\_drop\_packets);

printf("mtu\_drop\_packets = %d\n",(WP\_U32)flow\_stats.mtu\_drop\_packets);

printf("ttl\_drop\_packets = %d\n",(WP\_U32)flow\_stats.ttl\_drop\_packets);

printf("tx\_queue\_drop\_packets = %d\n",(WP\_U32)flow\_stats.tx\_queue\_drop\_packets);

printf("mpls\_drop = %d\n",(WP\_U32)flow\_stats.mpls\_drop);

printf("denied\_packets = %d\n",(WP\_U32)flow\_stats.denied\_packets);

printf("group\_filtered\_packets = %d\n",(WP\_U32)flow\_stats.group\_filtered\_packets);

printf("forwarded\_bytes = %d\n",(WP\_U32)flow\_stats.forwarded\_bytes);

printf("gtp\_bad\_headers = %d\n",(WP\_U32)flow\_stats.gtp\_bad\_headers);

printf("policer\_non\_conforming\_packets; = %d\n",(WP\_U32)flow\_stats.policer\_non\_conforming\_packets);

printf("-------------------------------\n");

printf("Bridge2 Port Statistics TDM %d\n",ii);

printf("-------------------------------\n");

memset(&bport\_stats, 0, sizeof(bport\_stats));

status = WP\_IwPortStatistics(h\_bridge2\_bport\_tdm[ii], &bport\_stats);

App\_TerminateOnError(status, "BportStatistics");

printf("rx\_valid\_packets = %d\n",(WP\_U32)bport\_stats.rx\_valid\_packets);

printf("discard\_vlan\_acceptable\_filter = %d\n",(WP\_U32)bport\_stats.discard\_vlan\_acceptable\_filter);

printf("discard\_ingress\_filter = %d\n",(WP\_U32)bport\_stats.discard\_ingress\_filter);

printf("discard\_bridge\_classifier = %d\n",(WP\_U32)bport\_stats.discard\_bridge\_classifier);

printf("discard\_unk\_macsa = %d\n",(WP\_U32)bport\_stats.discard\_unk\_macsa);

printf("deny\_mac\_sa = %d\n",(WP\_U32)bport\_stats.deny\_mac\_sa);

printf("deny\_mac\_da = %d\n",(WP\_U32)bport\_stats.deny\_mac\_da);

printf("rx\_bc\_valid\_packets = %d\n",(WP\_U32)bport\_stats.rx\_bc\_valid\_packets);

printf("rx\_mc\_valid\_packets = %d\n",(WP\_U32)bport\_stats.rx\_mc\_valid\_packets);

printf("forwarded\_uc\_packets = %d\n",(WP\_U32)bport\_stats.forwarded\_uc\_packets);

printf("forwarded\_bc\_packets = %d\n",(WP\_U32)bport\_stats.forwarded\_bc\_packets);

printf("forwarded\_mc\_packets = %d\n",(WP\_U32)bport\_stats.forwarded\_mc\_packets);

printf("protocol\_error = %d\n",(WP\_U32)bport\_stats.protocol\_error);

}

printf("-------------------------------\n");

printf(" Flow statistics LCP\n");

printf("-------------------------------\n");

memset(&flow\_stats,0,sizeof(flow\_stats));

status = WP\_IwFlowStatistics(h\_flow\_agg\_lcp,WP\_IW\_FLOW\_STAT,&flow\_stats);

App\_TerminateOnError(status, "WP\_IwFlowStatistics");

printf("forward\_packet = %d\n",(WP\_U32)flow\_stats.forward\_packet);

printf("fbp\_drop\_packets = %d\n",(WP\_U32)flow\_stats.fbp\_drop\_packets);

printf("mtu\_drop\_packets = %d\n",(WP\_U32)flow\_stats.mtu\_drop\_packets);

printf("ttl\_drop\_packets = %d\n",(WP\_U32)flow\_stats.ttl\_drop\_packets);

printf("tx\_queue\_drop\_packets = %d\n",(WP\_U32)flow\_stats.tx\_queue\_drop\_packets);

printf("mpls\_drop = %d\n",(WP\_U32)flow\_stats.mpls\_drop);

printf("denied\_packets = %d\n",(WP\_U32)flow\_stats.denied\_packets);

printf("group\_filtered\_packets = %d\n",(WP\_U32)flow\_stats.group\_filtered\_packets);

printf("forwarded\_bytes = %d\n",(WP\_U32)flow\_stats.forwarded\_bytes);

printf("gtp\_bad\_headers = %d\n",(WP\_U32)flow\_stats.gtp\_bad\_headers);

printf("policer\_non\_conforming\_packets; = %d\n",(WP\_U32)flow\_stats.policer\_non\_conforming\_packets);

printf("-------------------------------\n");

printf(" Flow statistics Cisco Control Packets\n");

printf("-------------------------------\n");

memset(&flow\_stats,0,sizeof(flow\_stats));

status = WP\_IwFlowStatistics(h\_flow\_agg\_cisco\_control,WP\_IW\_FLOW\_STAT,&flow\_stats);

App\_TerminateOnError(status, "WP\_IwFlowStatistics");

printf("forward\_packet = %d\n",(WP\_U32)flow\_stats.forward\_packet);

printf("fbp\_drop\_packets = %d\n",(WP\_U32)flow\_stats.fbp\_drop\_packets);

printf("mtu\_drop\_packets = %d\n",(WP\_U32)flow\_stats.mtu\_drop\_packets);

printf("ttl\_drop\_packets = %d\n",(WP\_U32)flow\_stats.ttl\_drop\_packets);

printf("tx\_queue\_drop\_packets = %d\n",(WP\_U32)flow\_stats.tx\_queue\_drop\_packets);

printf("mpls\_drop = %d\n",(WP\_U32)flow\_stats.mpls\_drop);

printf("denied\_packets = %d\n",(WP\_U32)flow\_stats.denied\_packets);

printf("group\_filtered\_packets = %d\n",(WP\_U32)flow\_stats.group\_filtered\_packets);

printf("forwarded\_bytes = %d\n",(WP\_U32)flow\_stats.forwarded\_bytes);

printf("gtp\_bad\_headers = %d\n",(WP\_U32)flow\_stats.gtp\_bad\_headers);

printf("policer\_non\_conforming\_packets; = %d\n",(WP\_U32)flow\_stats.policer\_non\_conforming\_packets);

printf("-------------------------------\n");

printf(" Flow statistics ENET2\n");

printf("-------------------------------\n");

memset(&flow\_stats,0,sizeof(flow\_stats));

status = WP\_IwFlowStatistics(h\_flow\_agg\_enet2,WP\_IW\_FLOW\_STAT,&flow\_stats);

App\_TerminateOnError(status, "WP\_IwFlowStatistics");

printf("forward\_packet = %d\n",(WP\_U32)flow\_stats.forward\_packet);

printf("fbp\_drop\_packets = %d\n",(WP\_U32)flow\_stats.fbp\_drop\_packets);

printf("mtu\_drop\_packets = %d\n",(WP\_U32)flow\_stats.mtu\_drop\_packets);

printf("ttl\_drop\_packets = %d\n",(WP\_U32)flow\_stats.ttl\_drop\_packets);

printf("tx\_queue\_drop\_packets = %d\n",(WP\_U32)flow\_stats.tx\_queue\_drop\_packets);

printf("mpls\_drop = %d\n",(WP\_U32)flow\_stats.mpls\_drop);

printf("denied\_packets = %d\n",(WP\_U32)flow\_stats.denied\_packets);

printf("group\_filtered\_packets = %d\n",(WP\_U32)flow\_stats.group\_filtered\_packets);

printf("forwarded\_bytes = %d\n",(WP\_U32)flow\_stats.forwarded\_bytes);

printf("gtp\_bad\_headers = %d\n",(WP\_U32)flow\_stats.gtp\_bad\_headers);

printf("policer\_non\_conforming\_packets; = %d\n",(WP\_U32)flow\_stats.policer\_non\_conforming\_packets);

printf("-------------------------------\n");

printf(" Flow statisics Default Bridge \n");

printf("-------------------------------\n");

memset(&flow\_stats,0,sizeof(flow\_stats));

status = WP\_IwFlowStatistics(h\_bridge1\_default\_agg,WP\_IW\_FLOW\_STAT,&flow\_stats);

App\_TerminateOnError(status, "WP\_IwFlowStatistics");

printf("forward\_packet = %d\n",(WP\_U32)flow\_stats.forward\_packet);

printf("fbp\_drop\_packets = %d\n",(WP\_U32)flow\_stats.fbp\_drop\_packets);

printf("mtu\_drop\_packets = %d\n",(WP\_U32)flow\_stats.mtu\_drop\_packets);

printf("ttl\_drop\_packets = %d\n",(WP\_U32)flow\_stats.ttl\_drop\_packets);

printf("tx\_queue\_drop\_packets = %d\n",(WP\_U32)flow\_stats.tx\_queue\_drop\_packets);

printf("mpls\_drop = %d\n",(WP\_U32)flow\_stats.mpls\_drop);

printf("denied\_packets = %d\n",(WP\_U32)flow\_stats.denied\_packets);

printf("group\_filtered\_packets = %d\n",(WP\_U32)flow\_stats.group\_filtered\_packets);

printf("forwarded\_bytes = %d\n",(WP\_U32)flow\_stats.forwarded\_bytes);

printf("gtp\_bad\_headers = %d\n",(WP\_U32)flow\_stats.gtp\_bad\_headers);

printf("policer\_non\_conforming\_packets; = %d\n",(WP\_U32)flow\_stats.policer\_non\_conforming\_packets);

printf("-------------------------------\n");

printf(" Flow statisics Default Router \n");

printf("-------------------------------\n");

memset(&flow\_stats,0,sizeof(flow\_stats));

status = WP\_IwFlowStatistics(h\_bridge2\_default\_agg,WP\_IW\_FLOW\_STAT,&flow\_stats);

App\_TerminateOnError(status, "WP\_IwFlowStatistics");

printf("forward\_packet = %d\n",(WP\_U32)flow\_stats.forward\_packet);

printf("fbp\_drop\_packets = %d\n",(WP\_U32)flow\_stats.fbp\_drop\_packets);

printf("mtu\_drop\_packets = %d\n",(WP\_U32)flow\_stats.mtu\_drop\_packets);

printf("ttl\_drop\_packets = %d\n",(WP\_U32)flow\_stats.ttl\_drop\_packets);

printf("tx\_queue\_drop\_packets = %d\n",(WP\_U32)flow\_stats.tx\_queue\_drop\_packets);

printf("mpls\_drop = %d\n",(WP\_U32)flow\_stats.mpls\_drop);

printf("denied\_packets = %d\n",(WP\_U32)flow\_stats.denied\_packets);

printf("group\_filtered\_packets = %d\n",(WP\_U32)flow\_stats.group\_filtered\_packets);

printf("forwarded\_bytes = %d\n",(WP\_U32)flow\_stats.forwarded\_bytes);

printf("gtp\_bad\_headers = %d\n",(WP\_U32)flow\_stats.gtp\_bad\_headers);

printf("policer\_non\_conforming\_packets; = %d\n",(WP\_U32)flow\_stats.policer\_non\_conforming\_packets);

}

void App\_Debug(WP\_boolean debug)

{

WP\_flow\_class\_rule flow\_class\_rule;

WP\_flow\_class\_forwarding\_action forwarding\_action;

if(debug)

{

printf("Debug is on. Turning it OFF\n");

forwarding\_action.flow\_aggregation = h\_flow\_agg\_tdm[1];

debug\_on = WP\_FALSE;

}

else

{

printf("Debug is off. Turning it ON\n");

forwarding\_action.flow\_aggregation = h\_bridge1\_default\_agg;

debug\_on = WP\_TRUE;

}

flow\_class\_rule.match\_action = WP\_FLOW\_CLASS\_MODE\_FLOW\_AGG;

flow\_class\_rule.action\_info.forwarding\_action = forwarding\_action;

flow\_class\_rule.wred\_entry = 0;

status = WP\_IwFlowModify(h\_iw\_sys\_bridge1, h\_bridge1\_dfcflow1, WP\_IW\_MODIFY\_WRITE\_THROUGH,

WP\_IW\_FLOW\_MOD\_FLOW\_AGG, &flow\_class\_rule);

App\_TerminateOnError(status," WP\_IwFlowModify() DFC");

status = WP\_IwSystemBuild(h\_iw\_sys\_bridge1);

App\_TerminateOnError(status," WP\_IwSystemBuild()");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Common routine to select IP Address \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void insert\_ip\_header(WP\_U32 ip\_sa, WP\_U32 ip\_da, WP\_U32 payload\_size,

WP\_U32 protocol, WP\_U32 offset, WP\_U8 \*header)

{

WP\_U32 packet\_length;

WP\_U32 temp\_sum, i;

WP\_U32 checksum = 0;

WP\_U8 \*ptr\_ip\_data;

WP\_U8 \*start\_ip\_data;

ptr\_ip\_data= header + offset;

start\_ip\_data = ptr\_ip\_data;

/\* Reset IP header just in case it has garbage \*/

memset(ptr\_ip\_data,0,IP\_HEADER\_SIZE);

/\* IP Version 4, header length = 20 bytes \*/

ptr\_ip\_data[IP\_VER\_HLEN] = 0x45;

/\* Calculate new Packet length \*/

packet\_length = payload\_size + IP\_HEADER\_SIZE;

ptr\_ip\_data[IP\_LENGTH] = (WP\_U8)(packet\_length >> 8);

ptr\_ip\_data[IP\_LENGTH+1] = (WP\_U8)(packet\_length & 0xff);

/\* Time to Live \*/

ptr\_ip\_data[IP\_TTL] = 64;

/\* Protocol \*/

ptr\_ip\_data[IP\_PROTOCOL] = protocol;

/\* Get new ip source address \*/

ptr\_ip\_data[IP\_SA] = (WP\_U8)(ip\_sa >> 24);

ptr\_ip\_data[IP\_SA+1] = (WP\_U8)(ip\_sa >> 16);

ptr\_ip\_data[IP\_SA+2] = (WP\_U8)(ip\_sa >> 8);

ptr\_ip\_data[IP\_SA+3] = (WP\_U8)(ip\_sa & 0xff);

/\* Get new ip destination address \*/

ptr\_ip\_data[IP\_DA] = (WP\_U8)(ip\_da >> 24);

ptr\_ip\_data[IP\_DA+1] = (WP\_U8)(ip\_da >> 16);

ptr\_ip\_data[IP\_DA+2] = (WP\_U8)(ip\_da >> 8);

ptr\_ip\_data[IP\_DA+3] = (WP\_U8)(ip\_da & 0xff);

/\* Calculate header checksum \*/

for(i=0;i < (IP\_HEADER\_SIZE/2); i++)

{

temp\_sum = (ptr\_ip\_data[0] << 8 | ptr\_ip\_data[1]);

checksum += temp\_sum;

ptr\_ip\_data += 2;

}

while (checksum & 0xffff0000)

checksum = (checksum & 0xffff) + ((checksum & 0xffff0000) >> 16);

checksum = ~checksum;

/\* Save calculated checksum value \*/

start\_ip\_data[IP\_CHECKSUM] = checksum >> 8;

start\_ip\_data[IP\_CHECKSUM+1] = checksum & 0xff;

}

#if \_WT\_MODULE\_TEST\_ENABLE\_

#include "wt\_util.c" /\* WT\_Identify \*/

#endif

#if \_APP\_MODULE\_TEST\_COMMENT\_ENABLE\_

#pragma check\_tests common

#endif