2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

Doing the Impossible: Using Formal Verification on Packet Based Data Paths

Doug Smith

DOULOS

accellera SYSTEMS INITIATIVE

Using Formal on Packet Based Data Paths

- Introduction
- Modeling Packet Base Input
- Results
- Summary

Conventional Formal Wisdom

- Formal is great for …
  - Control paths
  - Combinational logic
  - Bug hunting
  - Interface protocols
  - Specific applications (e.g., register checking, CDC, etc.)
  - Etc.
- Formal is bad for …
  - Data paths
  - Complex arithmetic
  - Cyclic logic paths
  - Non-synthesizable blocks
  - Possibly deep state space analysis
  - And so on …

Packet Based Data Paths – Why Not?

		0								1								2								3							
Octet	**Bit**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
0	**0**	Version				IHL				DSCP						ECN		Total Length															
4	**32**	Identification																Flags			Fragment Offset												
8	**64**	Time To Live								Protocol								Header Checksum															
12	**96**	Source IP Address																															
16	**128**	Destination IP Address																															
20	**160**	Options (if IHL > 5)																															
24	**192**																																
28	**224**																																
32	**256**																																
36	**260**	Data																															
...	**...**																																
9036	**72288**																																

Ethernet Jumbo frames – 9000+ bytes

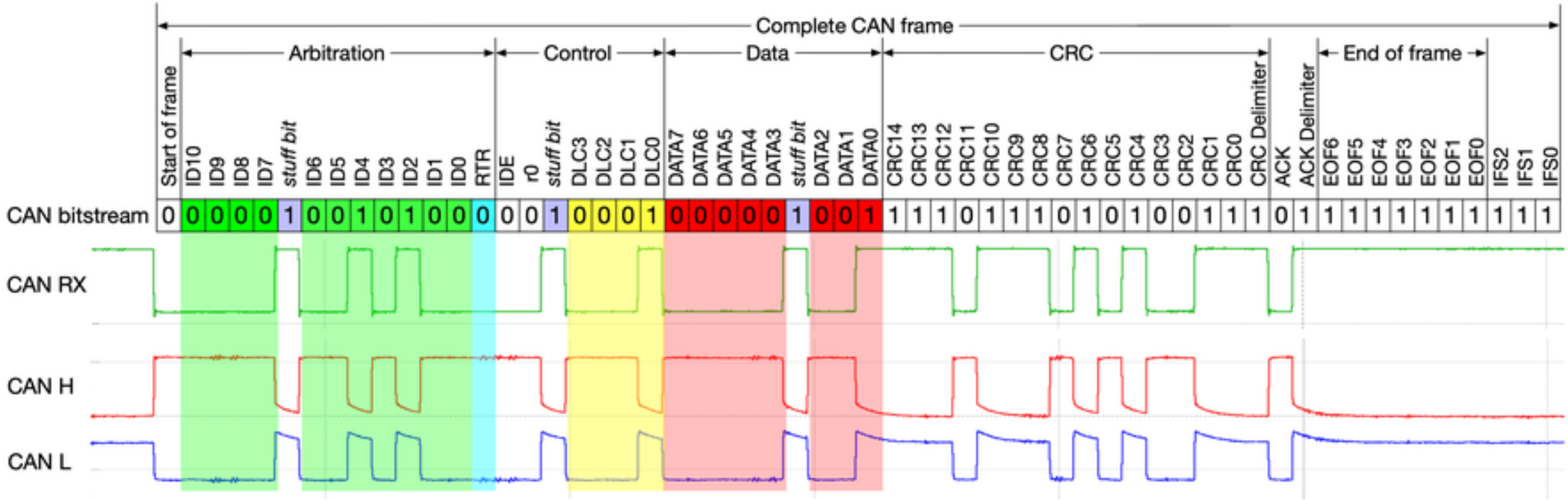
Too Much for Formal?

- 2^{72000+} state space = very large!
- But …
  - Legal state space is very small
  - Really checking correct parsing and embedded control information (i.e., control path)
  - No need to check data payload

		0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags				Fragment Offset											
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															
24	192																																
28	224																																
32	256																																
36	260	Data																															
…	…																																
9036	72288																																

Quite reasonable for formal!

Example



CAN Bus Protocol

Wikipedia, "CAN bus," [Online]. Available: https://en.wikipedia.org/wiki/CAN_bus. [Accessed 28 July 2022].

Using Formal on Packet Based Data Paths

Introduction

Modeling Packet Base Input

Results

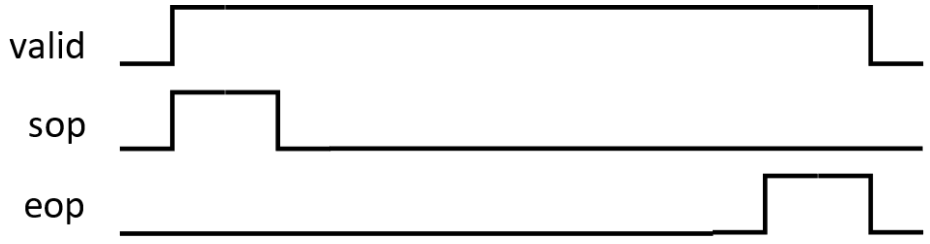
Summary

Model the Control Logic

Modeling code

```
bit       in_progress; // Transaction in progress
bit [7:0] total_bits;  // Total frame bits
bit [7:0] tx_bits;     // Frame bits transmitted

// Create an active flag
wire active = pkt_sof | in_progress;

always @(posedge clk or posedge rst) begin
    if ( rst )          in_progress <= 1'b0;
    else if ( pkt_eof ) in_progress <= 1'b0;
    else if ( pkt_sof ) in_progress <= 1'b1;
end
```

Typical protocol



Constraints

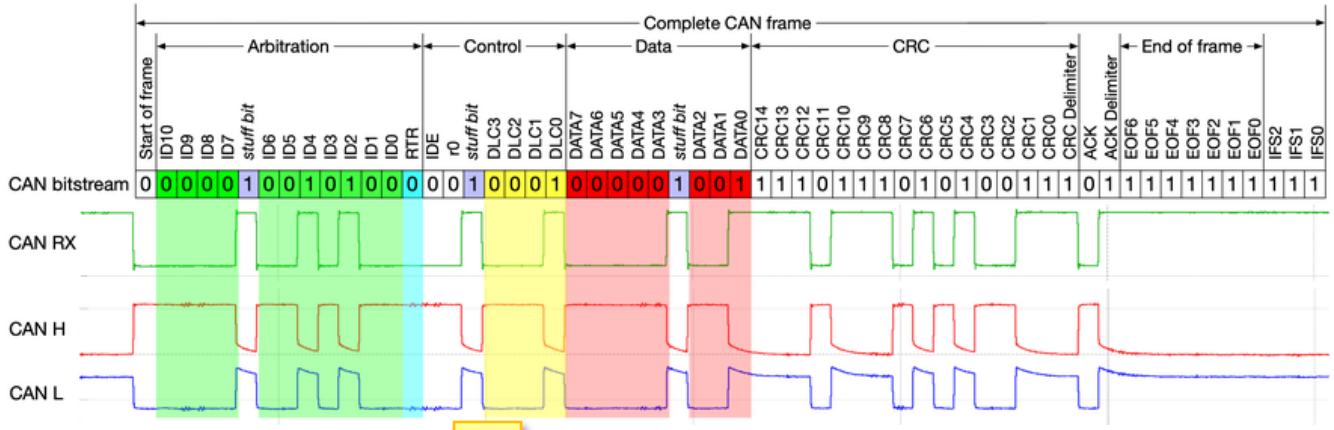
```
// Control signal constraints
asm_pkt_vld:    assume property ( active <-> pkt_vld );
asm_pkt_sof:    assume property ( $rose(pkt_vld) |-> $rose(pkt_sof) );
asm_pkt_eof:    assume property ( pkt_vld && (tx_bits >= total_bits) <-> pkt_eof );
asm_pkt_notsof: assume property ( in_progress |-> !pkt_sof );
```

Define the Packet Structure

Field name	Length (bits)
Start-of-frame	1
Identifier (green)	11
Stuff bit	1
Remote transmission request (RTR) (blue)	1
Identifier extension bit (IDE)	1
Reserved bit (r0)	1
Data length code (DLC) (yellow)	4
Data field (red)	0–64 (0-8 bytes)
CRC	15
CRC delimiter	1
ACK slot	1
ACK delimiter	1
End-of-frame (EOF)	7
Inter-frame spacing (IFS)	3

112 bits

2^{112} = Too big!




```
typedef struct packed {
    bit        ide;
    bit        r0;
    bit [3:0]  dlc;
    bit [1:0]  unused;
} control_t;
```

Break into 8, 16, or 32 bits

control_t

ide
r0
bit [3:0] dlc
bit [1:0] unused

Structs

```
// ------------------
// Start of frame
// ------------------
typedef struct packed {
   bit       sof;
   bit [6:0] unused;
} start_of_frame_t;


// ------------------
// Arbitration
// ------------------
typedef struct packed {
   bit [10:3] id;
} arbitration_high_t;

typedef struct packed {
   bit [2:0] id;
   bit       rtr;
   bit [3:0] unused;
} arbitration_low_t;
```

```
// ------------------
// Control
// ------------------
typedef struct packed {
   bit       ide;
   bit       r0;
   bit [3:0] dlc;
   bit [1:0] unused;
} control_t;


// ------------------
// Data
// ------------------
typedef struct packed {
   bit [7:0] value;
} data_t;
```

Each struct 8 bits

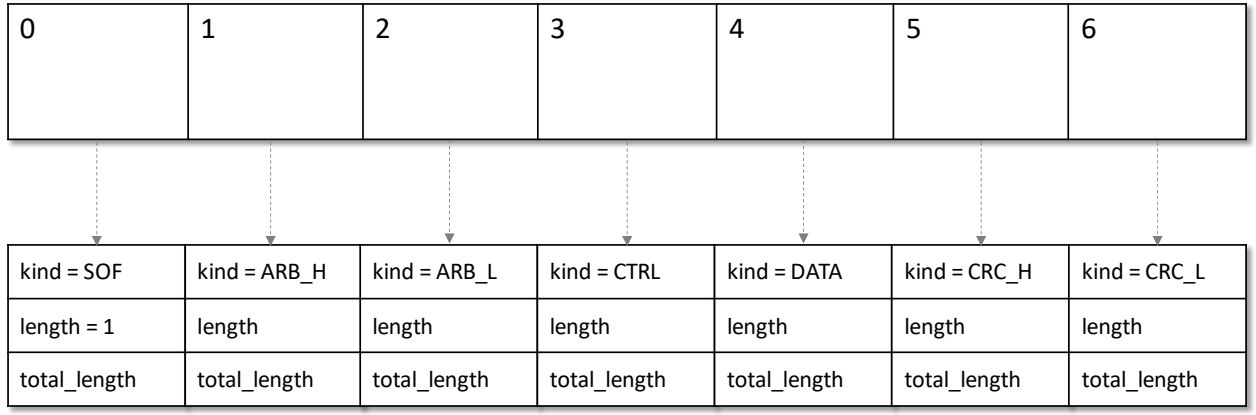
```
// ------------------
// CRC
// ------------------
typedef struct packed {
   bit [14: 7] crc;
} crc_high_t;

typedef struct packed {
   bit [6:0] crc;
   bit       unused;
} crc_low_t;

// Break out the CRC
// delimiter so no bit
// stuffing occurs
// during the delimiter
typedef struct packed {
   bit       crc_delim;
   bit [6:0] unused;
} crc_delimiter_t;
```

```
// ------------------
// Acknowledge
// ------------------
typedef struct packed {
   bit       ack;
   bit       ack_delim;
   bit [5:0] unused;
} ack_t;


// ------------------
// End of frame
// ------------------
typedef struct packed {
   bit [6:0] eof;
   bit       unused;
} end_of_frame_t;

typedef struct packed {
   bit [2:0] ifs;
   bit [4:0] unused;
} inter_frame_spacing_t;
```

Packet Arrays

```
// Combined packet type
typedef union packed {
   start_of_frame_t        sof;
   arbitration_high_t      arb_h;
   arbitration_low_t       arb_l;
   control_t               ctrl;
   data_t                  data;
   crc_high_t              crc_h;
   crc_low_t               crc_l;
   crc_delimiter_t         crc_delimiter;
   ack_t                   ack;
   end_of_frame_t          eof;
   inter_frame_spacing_t   ifs;
   bit [7:0]               qbits;
} pkt_item_t;

pkt_item_t packet [NUM_ITEMS+1];
```

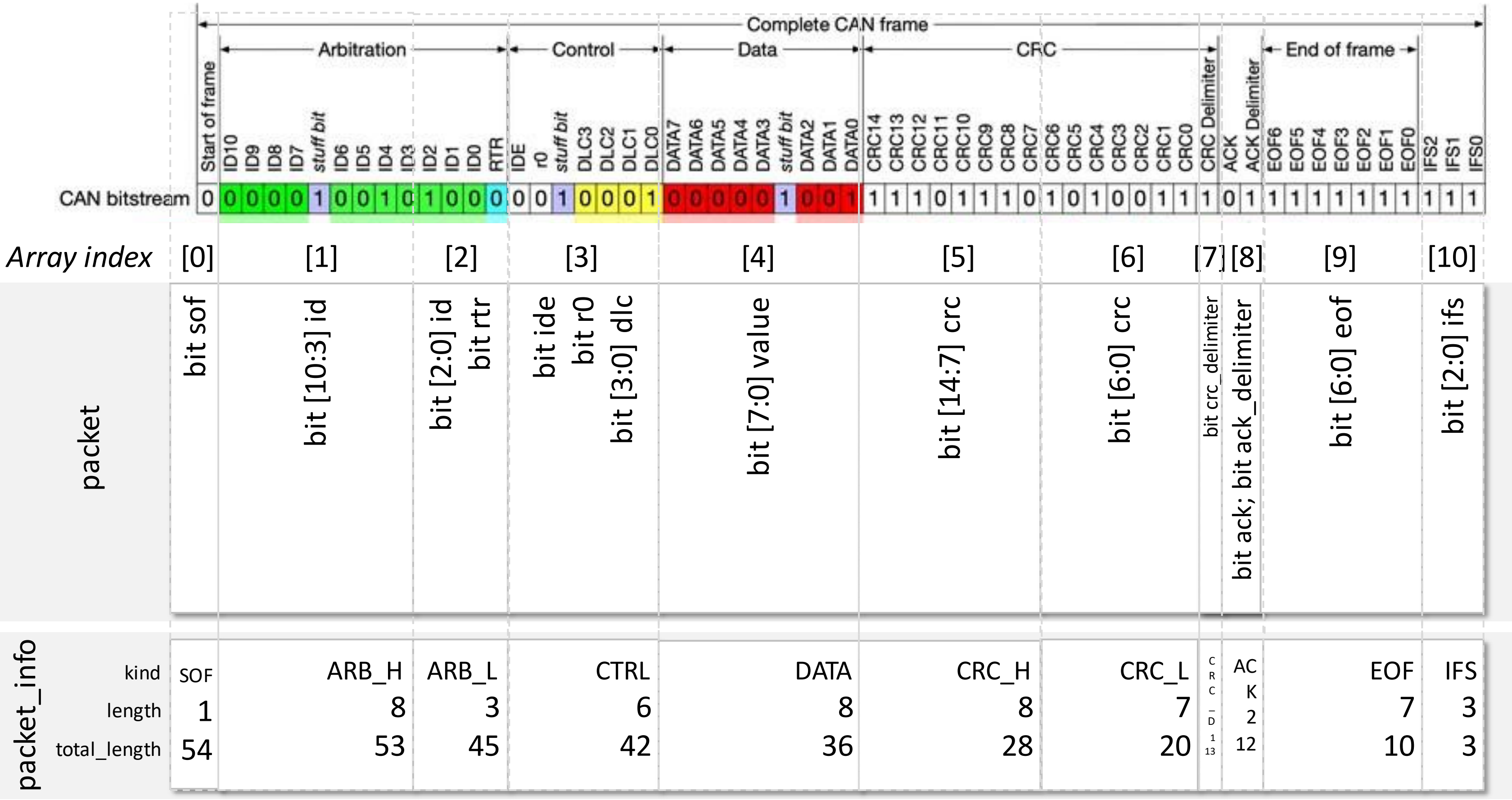
```
pkt_item_t packet [NUM_ITEMS+1];
```




```
packet_info_t packet_info [NUM_ITEMS+1];
```

```
typedef struct packed {
   pkt_item_kind_t   kind;
   bit [7:0]         length;
   bit [7:0]         total_length;
} packet_info_t;

packet_info_t packet_info [NUM_ITEMS+1];
```

Formal’s View of the Packet



Array index	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
packet	bit sof	bit [10:3] id	bit [2:0] id bit rtr	bit ide bit r0 bit [3:0] dlc	bit [7:0] value	bit [14:7] crc	bit [6:0] crc	bit crc_delimiter	bit ack; bit ack_delimiter	bit [6:0] eof	bit [2:0] ifs
packet_info kind	SOF	ARB_H	ARB_L	CTRL	DATA	CRC_H	CRC_L	CRC_D	ACK	EOF	IFS
length	1	8	3	6	8	8	7	1	2	7	3
total_length	54	53	45	42	36	28	20	13	12	10	3

Define the Packet Info Constraints

```
sequence seq_kind(n, k);
   packet_info[n].kind == k;
endsequence

sequence seq_total_length(n);
   packet_info[n].total_length == packet_info[n+1].total_length +
                                  packet_info[n].length;
endsequence

sequence seq_length(n, l);
   ( packet_info[n].length == l ) and seq_total_length(n);
endsequence

sequence seq_terminate_length(n);
   ( packet_info[n].length == 0 ) &&
   ( packet_info[n].total_length == 0 );
endsequence
```

```
seq_kind(0, SOF);
seq_kind(1, ARB_H);
seq_kind(2, ARB_L);
...
sequence seq_kind(n,k);
  packet_info[n].kind == k;
endsequence
```

packet_info_t

kind
length
total_length

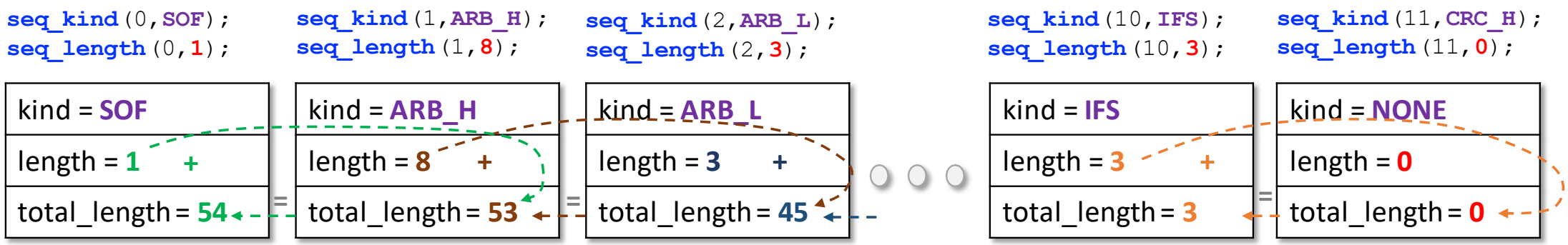
0	1	2	3	4	5	6
SOF	ARB_H	ARB_L	CTRL	DATA	CRC_H	...

```
packet_info_t packet_info [NUM_ITEMS+1];
```

In detail …

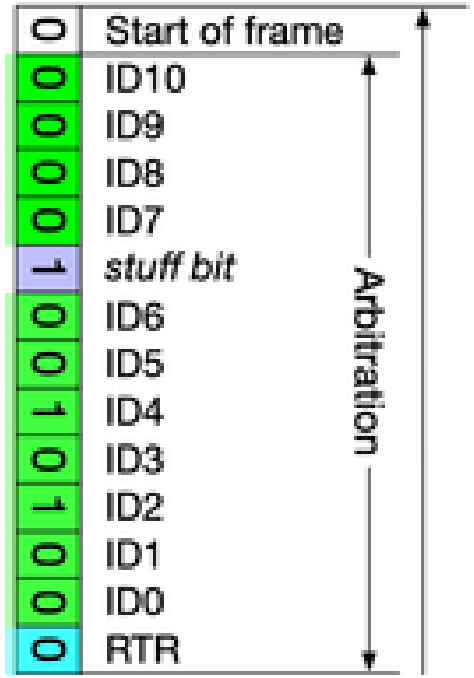
Reach into next element

```
sequence seq_total_length(n);
   packet_info[n].total_length == packet_info[n+1].total_length +
                                  packet_info[n].length;
endsequence

sequence seq_length(n, l);
   ( packet_info[n].length == l ) and seq_total_length(n);
endsequence
```



Define the Packet Constraints

```
//
// Start of frame
//
property prop_sof(n);
    seq_kind(n,SOF) and
    seq_length(n,1) and
    (packet[n].sof.sof == '0) and
    (packet[n].sof.unused == '0);
endproperty

//
// Arbitration
//
enum bit { DATA_FRAME    = 0, REMOTE_FRAME = 1 } frame_type;
bit [10:0] id;

property prop_arb_h(n);
    seq_kind(n,ARB_H) and
    seq_length(n,8) and
   (packet[n].arb_h.id == id[10:3]);
endproperty

property prop_arb_l(n);
    seq_kind(n,ARB_L) and
    seq_length(n,4) and
    (packet[n].arb_l.id == id[2:0]) and
    (packet[n].arb_l.rtr == frame_type) and
    (packet[n].arb_h.unused == '0);
endproperty
```

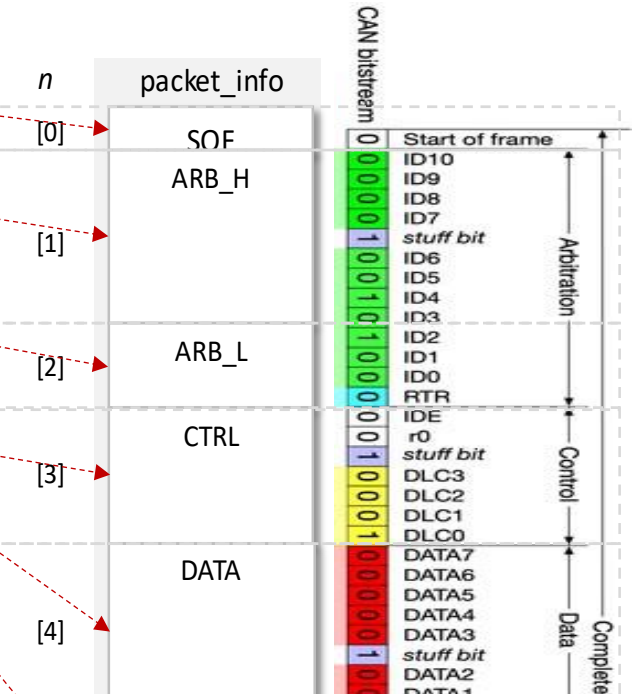


Define a Top Level Conditional Constraint

```
// ------------------------------------------------------------------
// Apply packet constraints
// ------------------------------------------------------------------
property prop_pkt(n);
        if ( packet_info[n].kind == SOF           ) prop_arb_h(n+1)
   else if ( packet_info[n].kind == ARB_H         ) prop_arb_l(n+1)
   else if ( packet_info[n].kind == ARB_L         ) prop_control(n+1)
   else if ( ( payload_size == 0    ) && ( packet_info[n].kind == CTRL) ) prop_crc_h(n+1)
   else if ( ( payload_size  > 0    ) && ( packet_info[n].kind == CTRL) ) prop_data(n+1)
   else if ( (( payload_size+3) >  n) && ( packet_info[n].kind == DATA )) prop_data(n+1)
   else if ( (( payload_size+3) <= n) && ( packet_info[n].kind == DATA )) prop_crc_h(n+1)
   else if ( packet_info[n].kind == CRC_H         ) prop_crc_l(n+1)
   else if ( packet_info[n].kind == CRC_L         ) prop_crc_delimiter(n+1)
   else if ( packet_info[n].kind == CRC_DELIMITER ) prop_ack(n+1)
   else if ( packet_info[n].kind == ACK           ) prop_eof(n+1)
   else if ( packet_info[n].kind == EOF           ) prop_ifs(n+1)
   else                                             prop_none(n+1);
endproperty
```

Constraining to Next Element

```
assume property (pkt_sof |-> prop_sof(0));

property prop_pkt(n);
      if ( packet_info[0].kind == SOF ) prop_arb_h(0+1)

 else if ( packet_info[1].kind == ARB_H ) prop_arb_l(1+1)

 else if ( packet_info[2].kind == ARB_L ) prop_control(2+1)

 else if ( ( payload_size > 0 ) && ( packet_info[3].kind == CTRL) ) prop_data(3+1)

 else if ( (( payload_size+3) <=4) && ( packet_info[4].kind == DATA )) prop_crc_h(4+1)

 ...
```



Generate the Packet

```
// Initialize the packet
asm_pkt_init_start    : assume property ( pkt_sof |-> prop_sof(0) );
asm_pkt_init_last     : assume property ( prop_none(NUM_ITEMS) );
```

```
// Create the packet/frame
for (genvar i = 0; i < NUM_ITEMS-1; i++ )
begin : pkt_init
   asm_init_pkt : assume property ( pkt_sof |-> prop_pkt(i) );
end
```

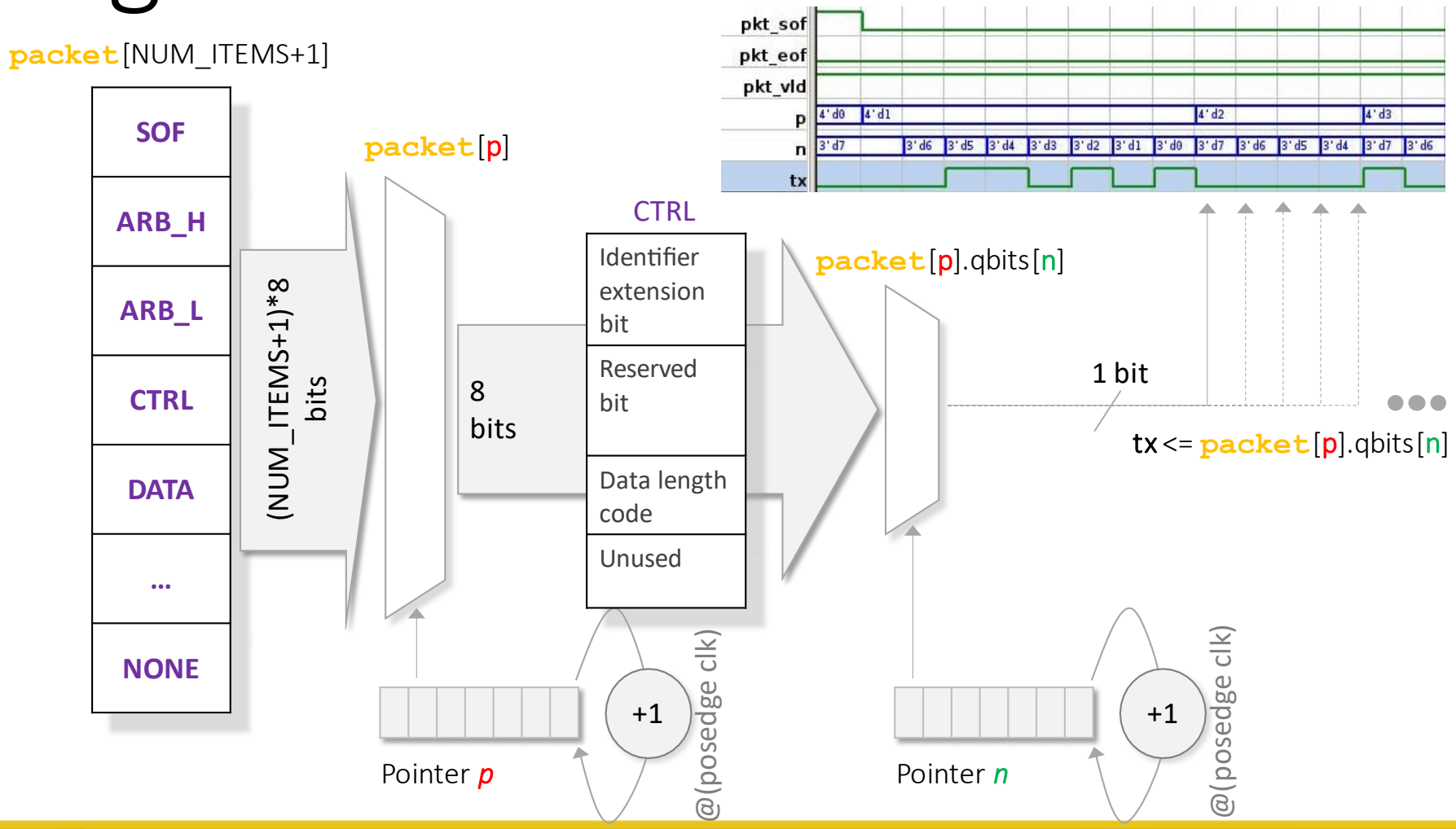
```
// Generate the CRC
asm_calc_crc: assume property ( pkt_vld |-> crc == calc_crc() && $stable(crc) );
```

Model Packet Driving Logic

```
// ---------------------------------------------
// Model the packet driving logic
// ---------------------------------------------
bit       tx_out;              // Data to drive
                               // to the output

bit [3:0] p;
bit [2:0] n;

always @(posedge clk or posedge rst)
begin
   if ( rst ) begin
      tx_out   <= '0;
      p        <= '0;
      n        <= $bits(pkt_item_t) - 1'b1;
      tx_bits  <= '0;
   end
   else begin

   // Transfer the data
   if ( pkt_vld ) begin
         // Drive the packet data
         tx_out <= packet[p].qbits[n];
```

```
         // Keep track of how many bits sent
         tx_bits <= tx_bits + 1'b1;

         // Update pointers
         if ( ( $bits(pkt_item_t) - n ) == packet_info[p].length ) begin
            if ( packet_info[p].kind == NONE )
               p <= '0;                      // Wrap back to beginning
            else
               p <= p + 1'b1;                // Next packet item
               n <= $bits(pkt_item_t) - 1'b1; // Start with the top bit
            end
            else begin
               // Next bit in the packet item
               n <= n - 1'b1;
            end
         end
         else begin
            // Clear when not valid
            p <= '0;
            n <= $bits(pkt_item_t) - 1'b1;
         end
      end
   end
```

Driving the Packet



Drive the Packet

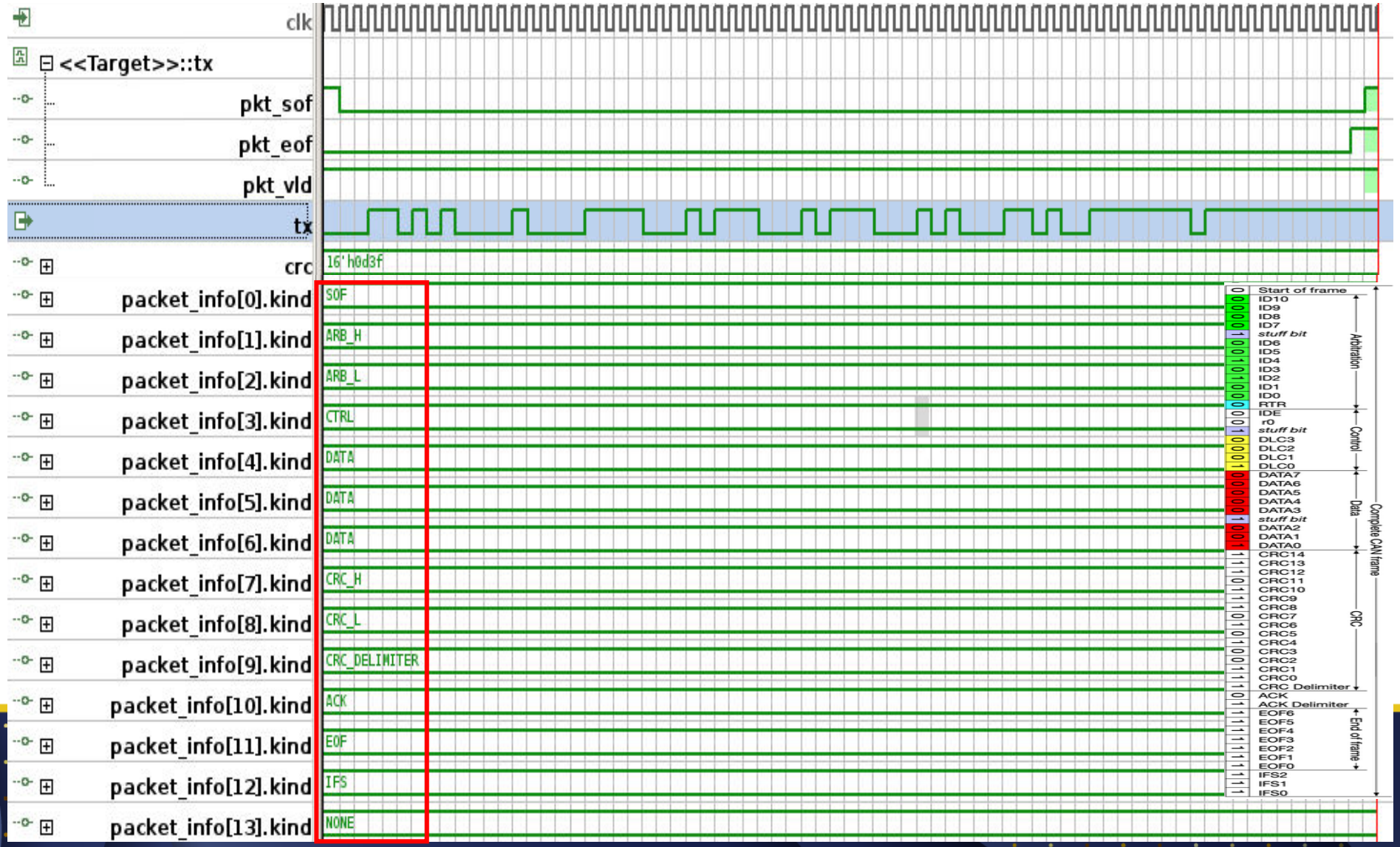
```
// Drive the frame
asm_drive_data      : assume property (  pkt_vld |-> tx == tx_out );
asm_undriven        : assume property ( !pkt_vld |-> tx == 1'b1 );

asm_pkt_stable      : assume property (  pkt_vld |-> $stable(packet) );
asm_pkt_info_stable: assume property (  pkt_vld |-> $stable(packet_info) );
```

Check the Packet

```
// Constrain control signals
property prop_transfer;
   pkt_sof <-> pkt_vld[*1:$] ##0 pkt_eof;
endproperty
```

```
// Generate waveform of frame
cov_gen_packet:  cover property ( prop_transfer );
```

Waveform



Error Injection

```
// Used by formal tool to select
// good or bad packets
enum bit  { FALSE = 0,
            TRUE = 1 } pkt_good;
```

```
property prop_sof(n);
   seq_kind(n,SOF) and
   seq_length(n,1) and
   (packet[n].sof.sof == '0) and
   (packet[n].sof.unused == '0);
endproperty
```

```
property prop_sof(n);
   (packet[n].sof.sof == '0) and
   (packet[n].sof.unused == '0);
endproperty

// Add new property
property prop_pkt_sof(n);
   seq_kind(n,SOF) and
   seq_length(n,1) and
   if ( pkt_good )      prop_sof(n)
   else                 not(prop_sof(n));
endproperty
```

Using Formal on Packet Based Data Paths

Introduction

Modeling Packet Base Input

Results

Summary

Results

Design	Development Time	Compile Time	Verify Time	Results
Ethernet *(TCP/IP with Jumbo frames)*	1 month	1 minute or less	3 – 6 minutes for a waveform 3 – 30 minutes for proof	A number of design bugs found
CAN Bus *(Base frame with bit stuffing)*	1 week	20 – 60 seconds	Seconds for waveform Minutes for proof	*Demonstration purposes*

Quick Summary of Steps

Model the control logic

Define the packet structure

Define the packet constraints

Apply the packet constraints

Generate the packet

Model the packet driving logic

Check the packets

Download source code here:
https://www.doulos.com/Formal4PBDP

Questions?