SJTAG and P1687.1 interoperation: Options and Issues

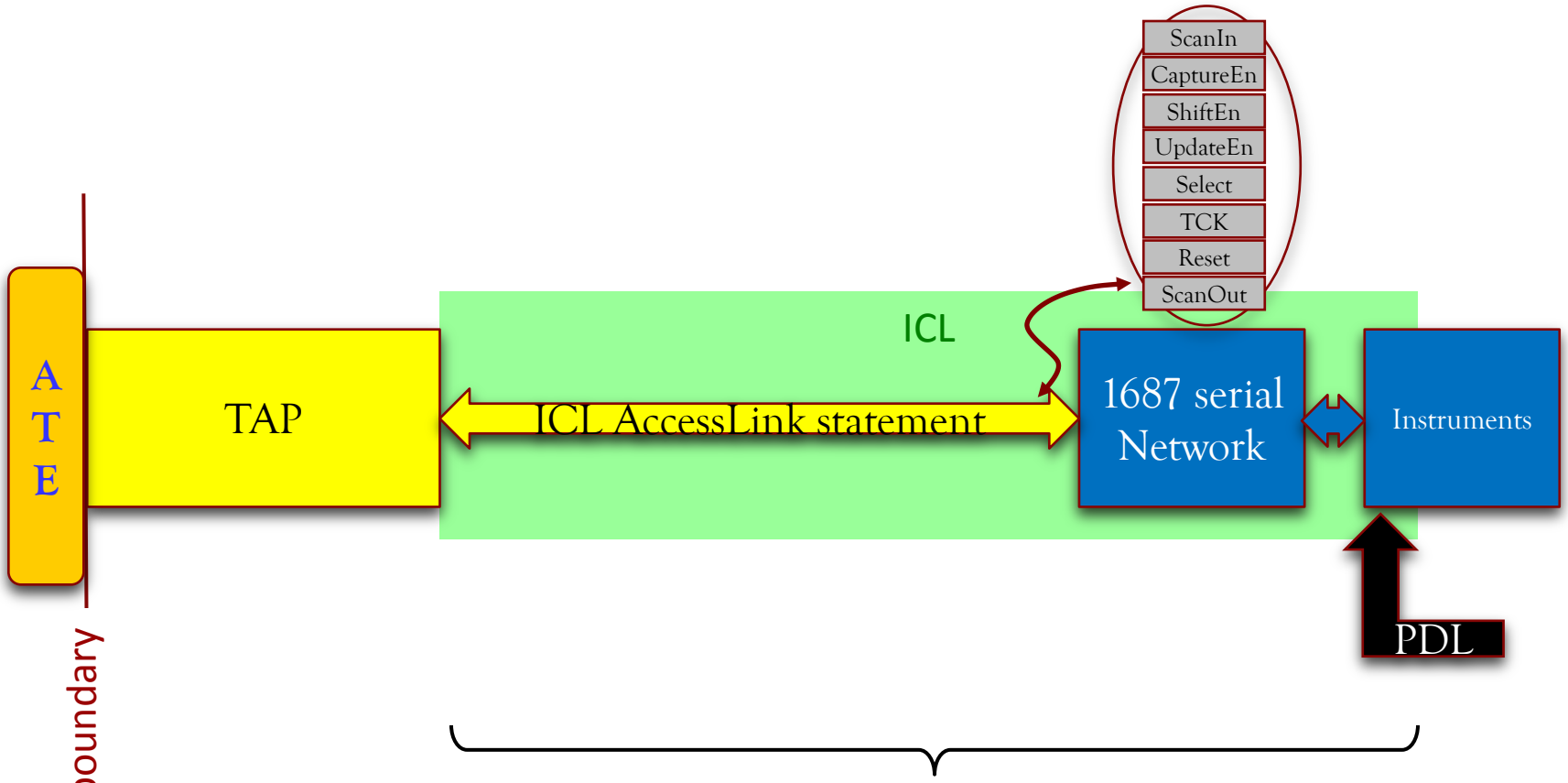
Jeff Rearick

August 28, 2017

Outline:

- Slides 2-5: 1687 and P1687.1 working proposal
- Slide 6: Simple SJTAG / P1687.1 distinction
- Slides 7-9: However, … (P1687.1 suggestion from Michele and Brad)

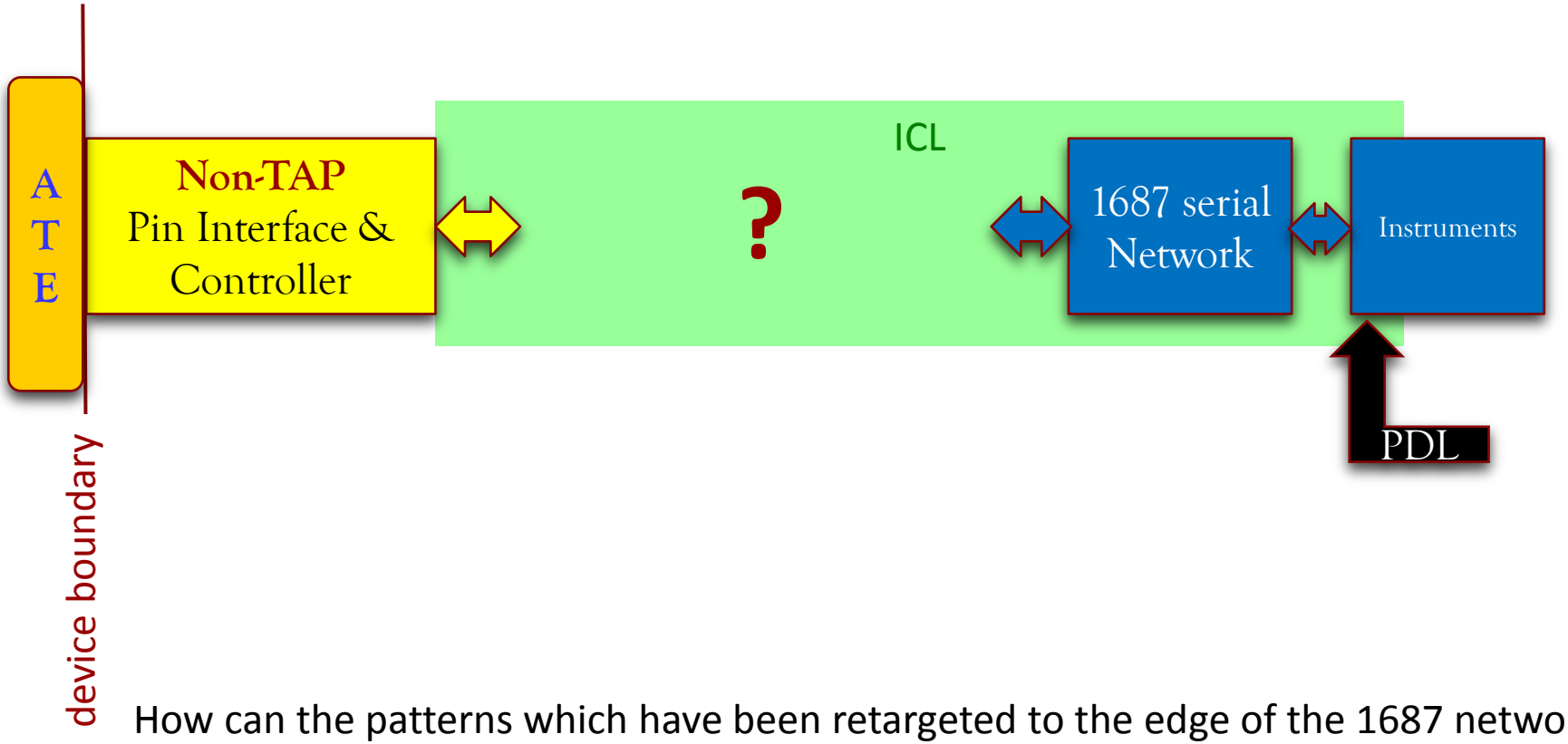
1687 serial access model



P1687 ICL describes 3 items:

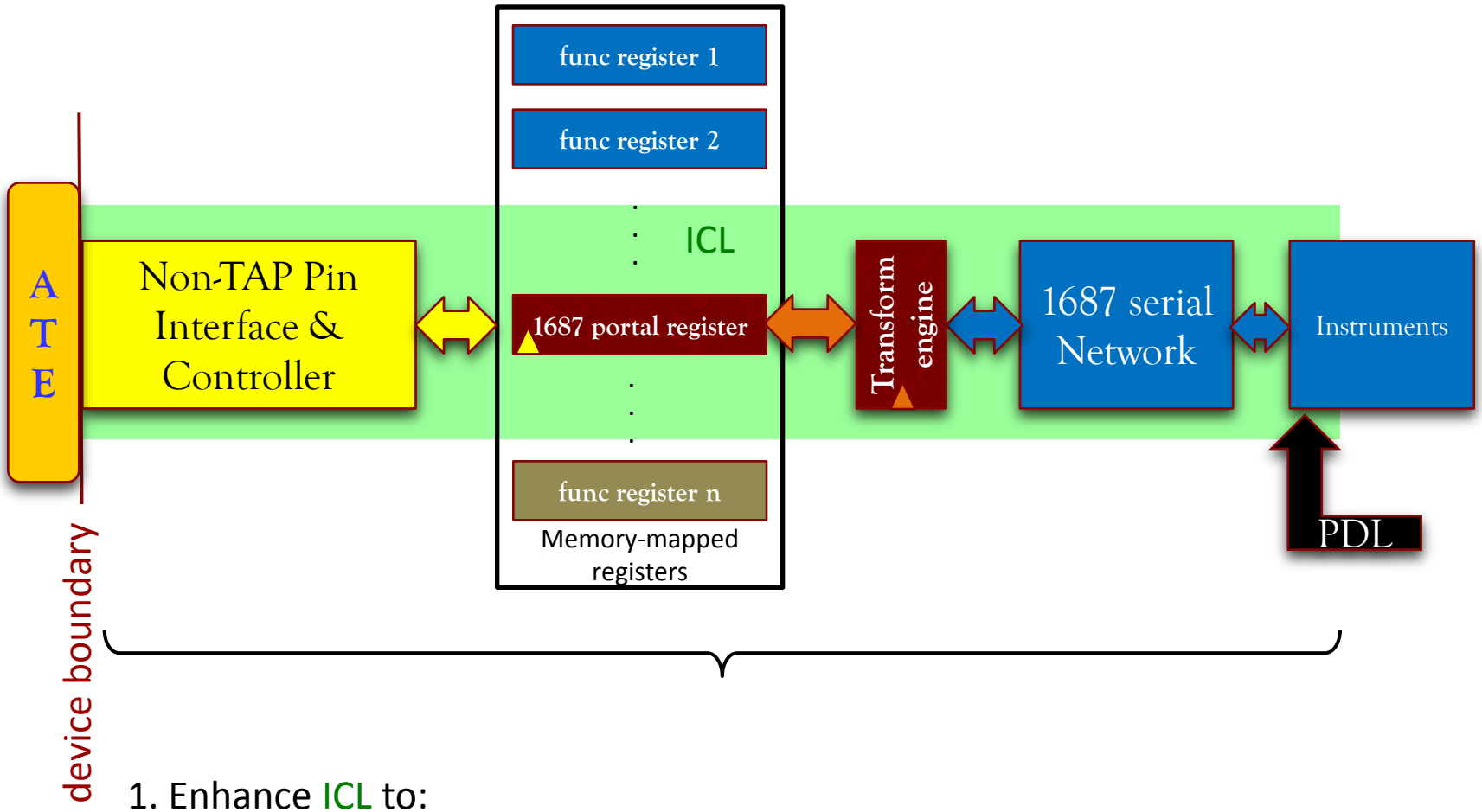
- Instrument ports & registers (where PDL is written)
- 1687 serial network (through which retargeting is done)
- AccessLink to TAP
  - Identifies instruction to activate serial network
  - Points to BSDL

P1687.1 problem statement



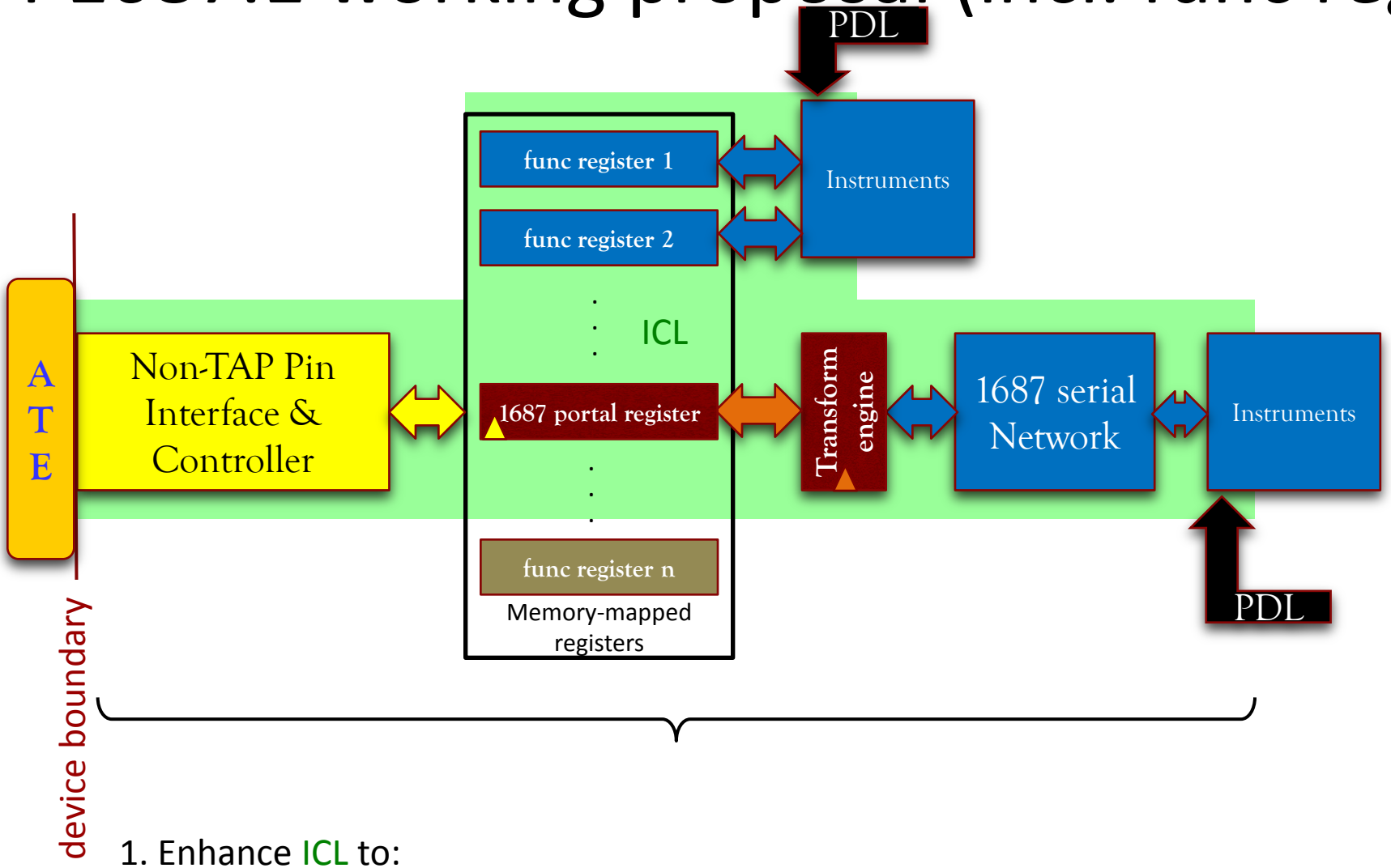
How can the patterns which have been retargeted to the edge of the 1687 network be translated to a non-TAP interface?

P1687.1 working proposal



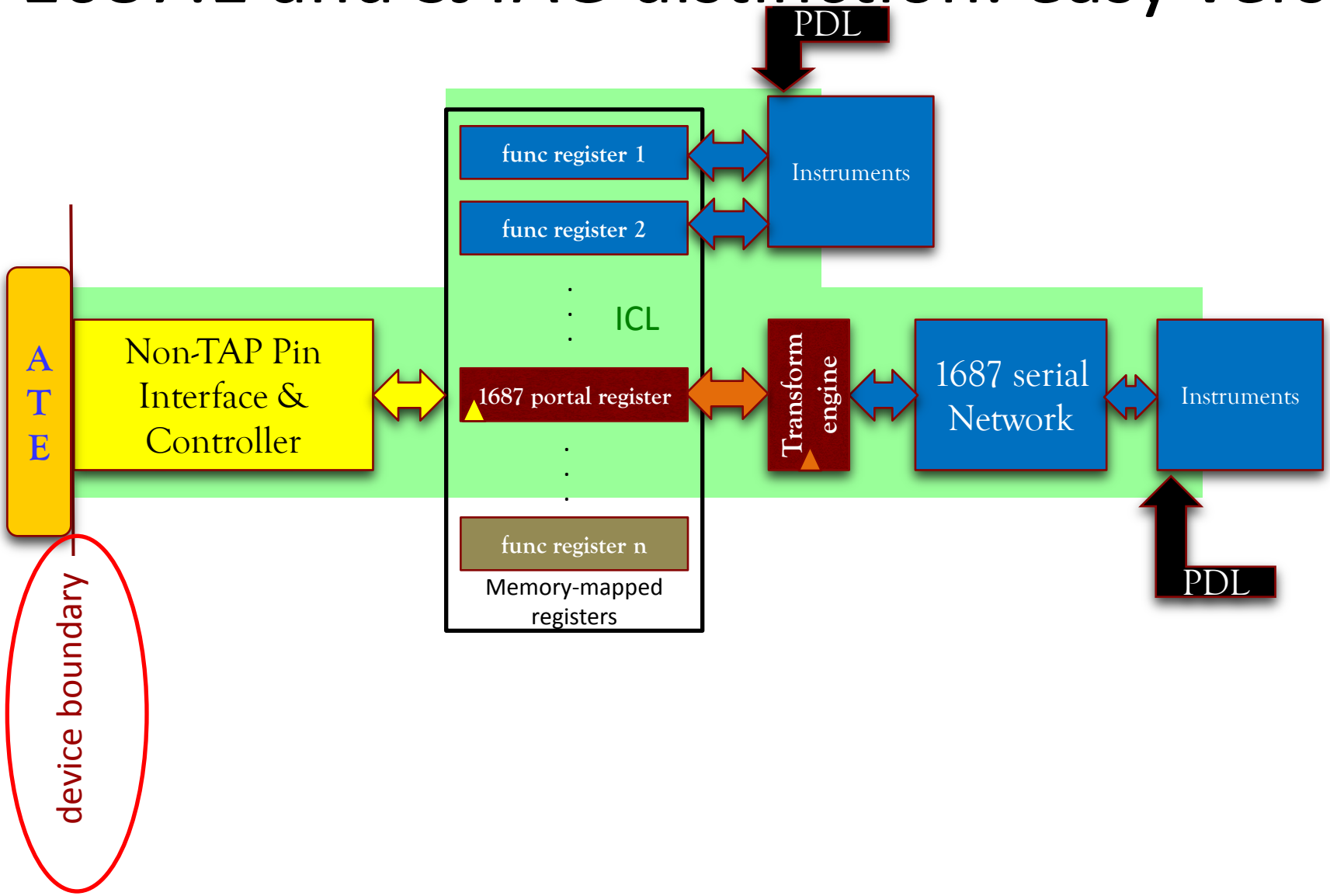
1. Enhance ICL to:
   a. Refer to our *TransformEngine* (implicitly; fully specify the TE in the standard)
   b. Allow pin-based write/read callbacks for addressable DataRegister (*portal register*)
2. Enhance PDL to allow "payload" tracking within callbacks via "iNote –payload"

P1687.1 working proposal (incl. func reg)



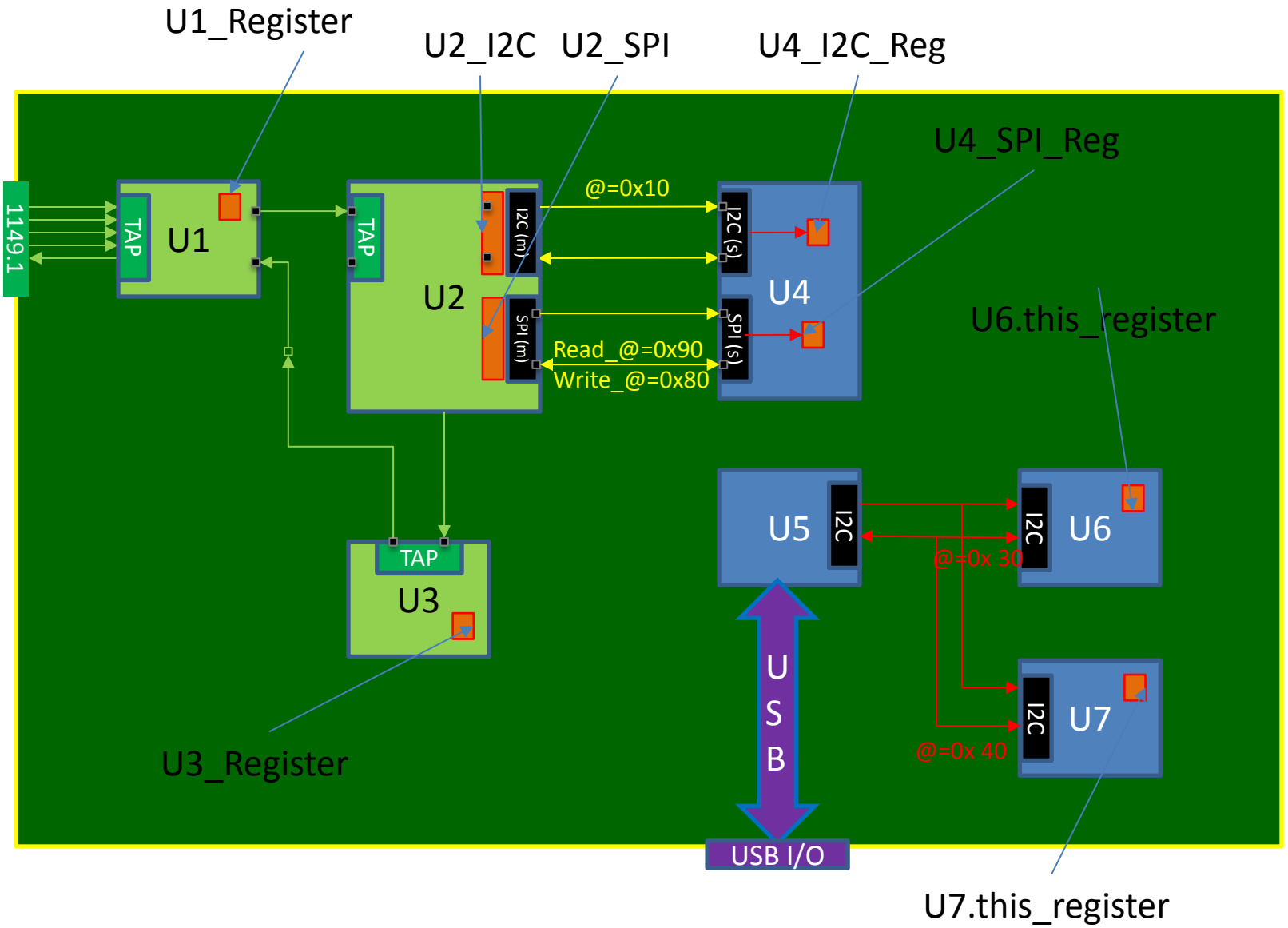
1. Enhance ICL to:
   a. Refer to our *TransformEngine* (implicitly; fully specify the TE in the standard)
   b. Allow pin-based write/read callbacks for addressable DataRegister (*portal register*)
2. Enhance PDL to allow "payload" tracking within callbacks via "iNote –payload"

P1687.1 and SJTAG distinction: easy version

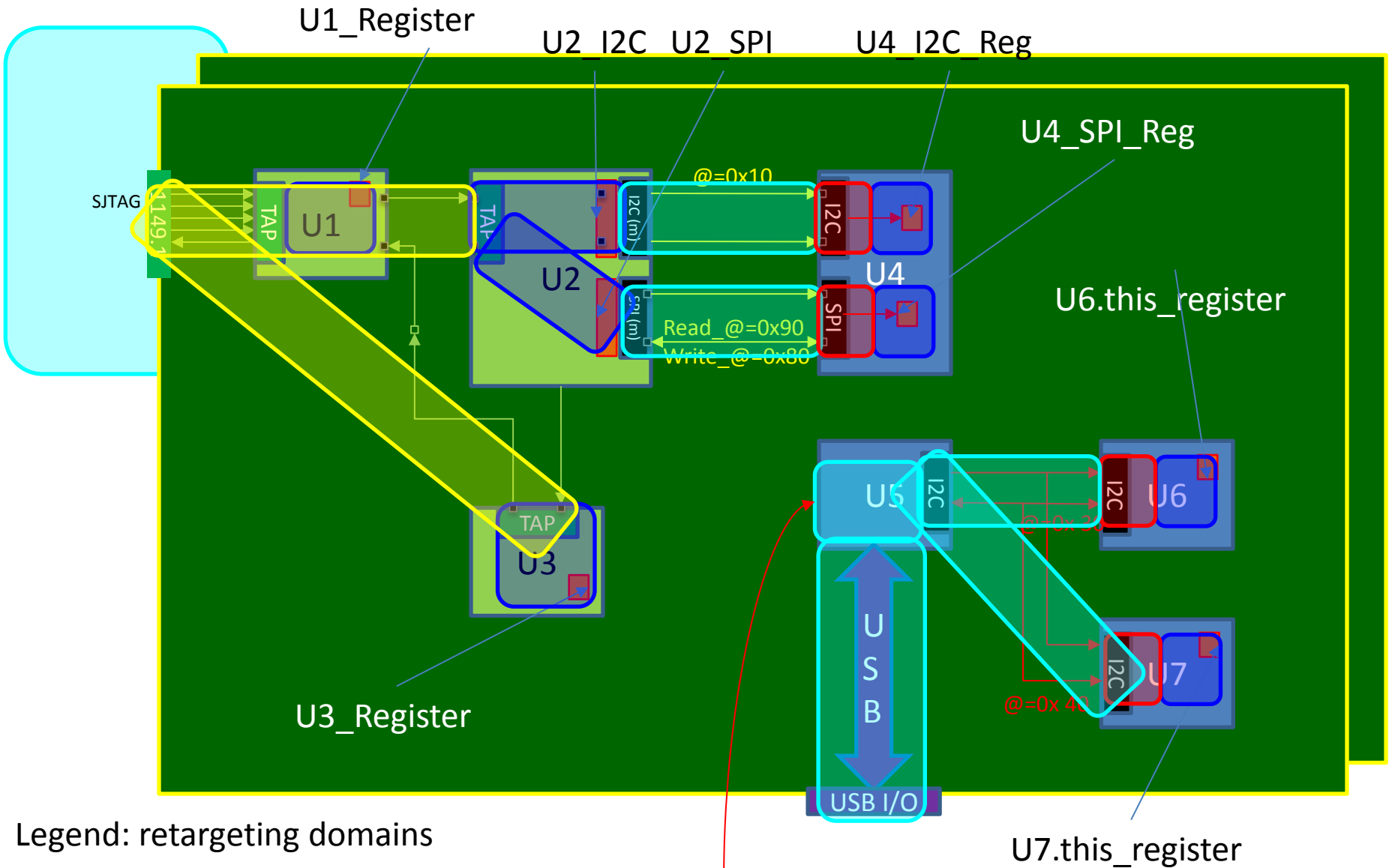


Observation: if the level of integration is a firm boundary (i.e. the device boundary), then P1687.1 is inside the device and SJTAG is outside the device. Simple. However….

Board example



Retargeting domains: Jeff's opinion



NEW

Board example for dfns

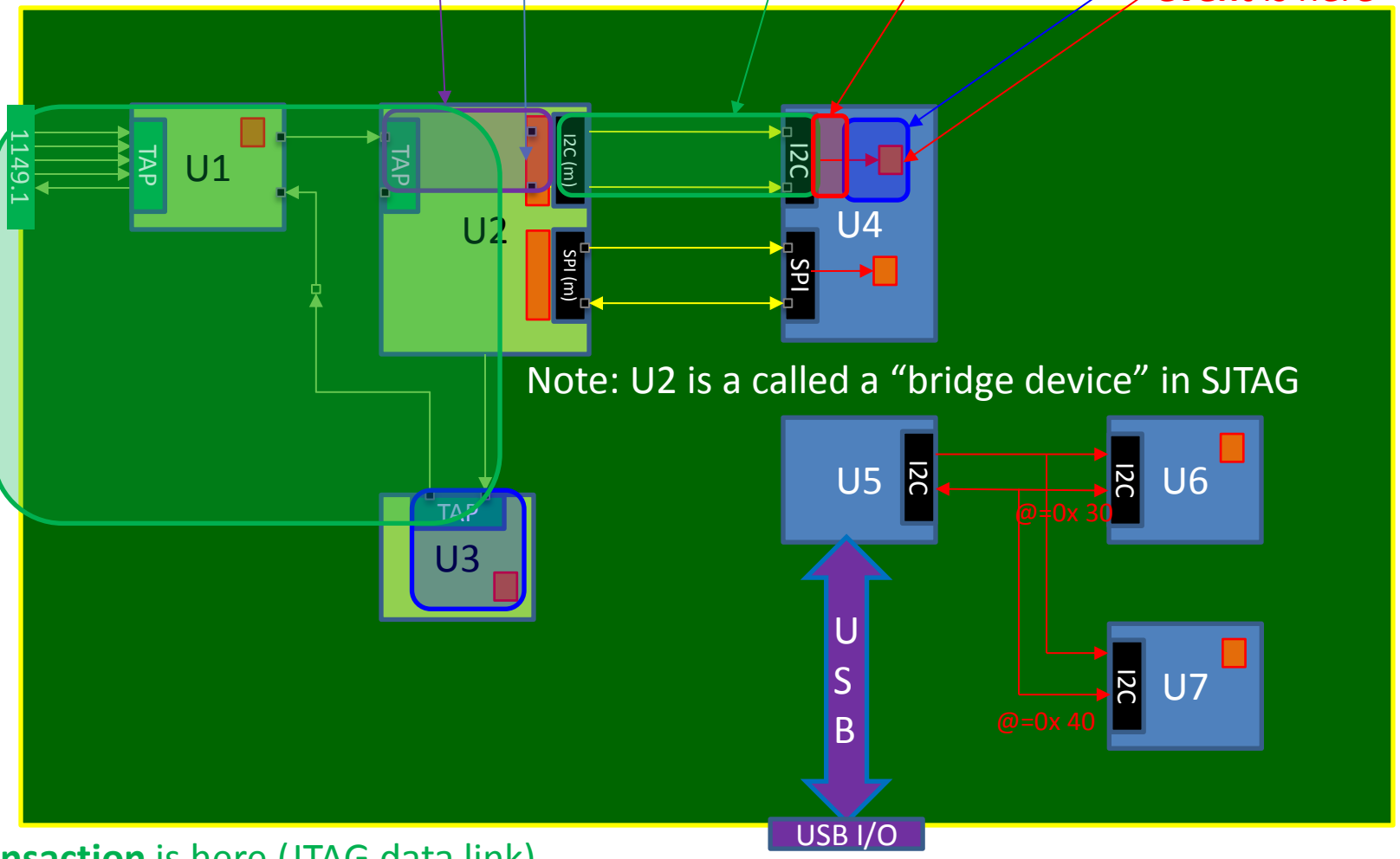
a **Transformation** happens here (AccessLink)

another **Transformation** happens here (access link from I2C Master to JTAG

[data] transaction is here (I2C data link)

Note: this may or may not be described in 1687; could be ad hoc [warning: automation issue here if ad hoc – perhaps the presence of ICL is what makes this a 1687-compatible system

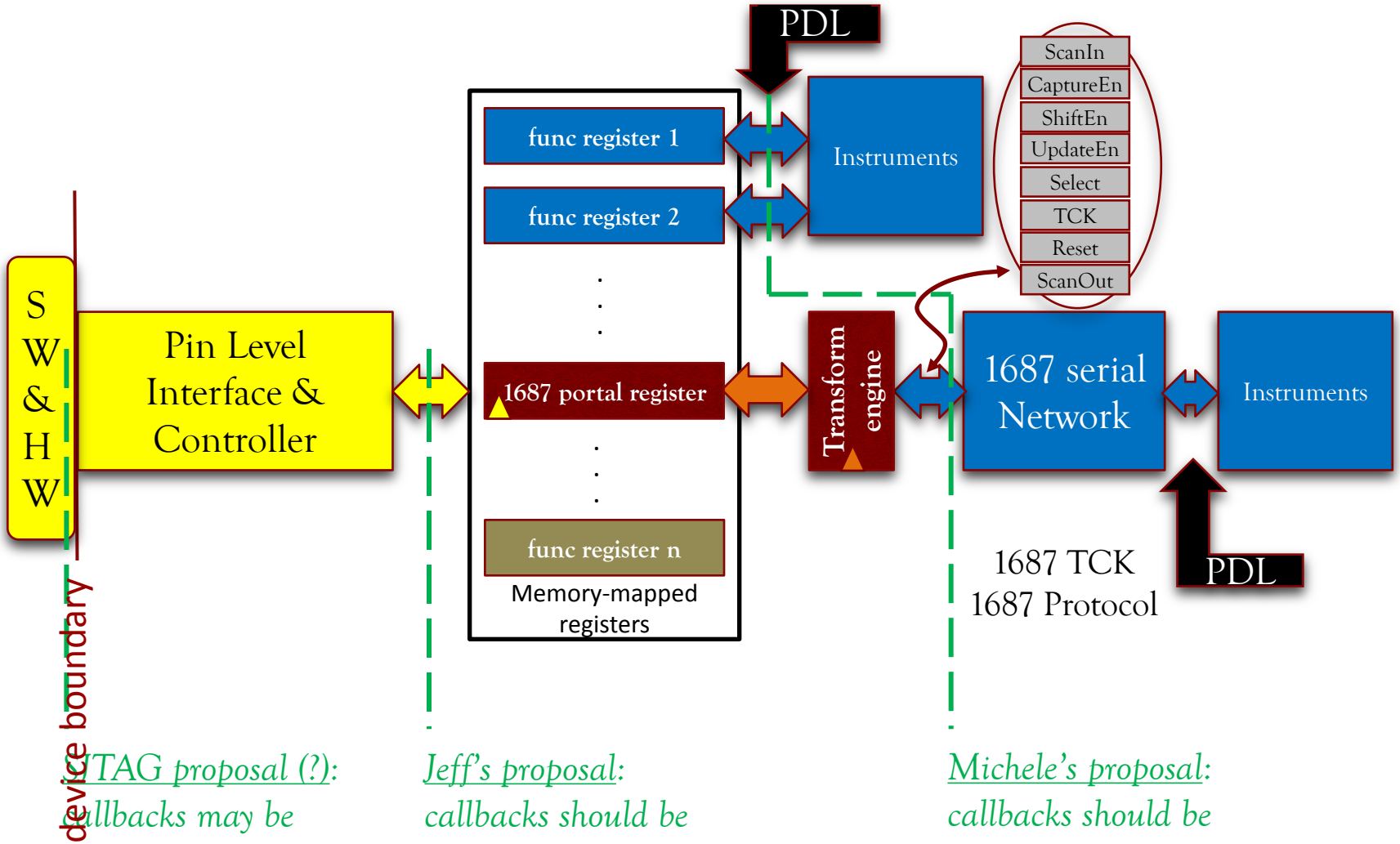
retargeting **event** is here



[data] transaction is here (JTAG data link)

The question for us to answer with SJTAG team: if our common goal is symbolic access to embedded instruments, and if the level of integration is not a firm boundary, then what is the unique mission of P1687.1 and what is the unique mission of SJTAG?

Possible 1687.1 callback retargeting endpoints



JTAG proposal (?): callbacks may be written at device interfaces (?)

Jeff's proposal: callbacks should be written for the portal register

… because we need to teach the community what to build (Portal register and Transform Engine)

Michele's proposal: callbacks should be written at the edge of 1687 network

… because every hop between the pins and the 1687 network can be extracted from the data sheet

Important insights

- Insight: there are two aspects of retargeting:
  - The **interface** (aka location or endpoint) to which a pattern is retargeted: this determines the *protocol* for the retargeted patterns
    - Edge of the 1687 network
    - 1687 portal register (including the sequences to get through the Transform Engine)
    - Device pin interface (TAP, I2C, SPI, …)
  - The **format** in which the retargeted pattern is delivered
    - PDL, SVF, verilog testbench, STIL, …
- Insight: there are two roles of callbacks
  - Specifying the mechanics of transporting data from one interface to another
  - Providing an information communication standard between IP providers and test integrators
- Realization: Payload tracking is essential
  - Retargeting MUST support diagnosis by linking the retargeted data bits at each interface hop to the source

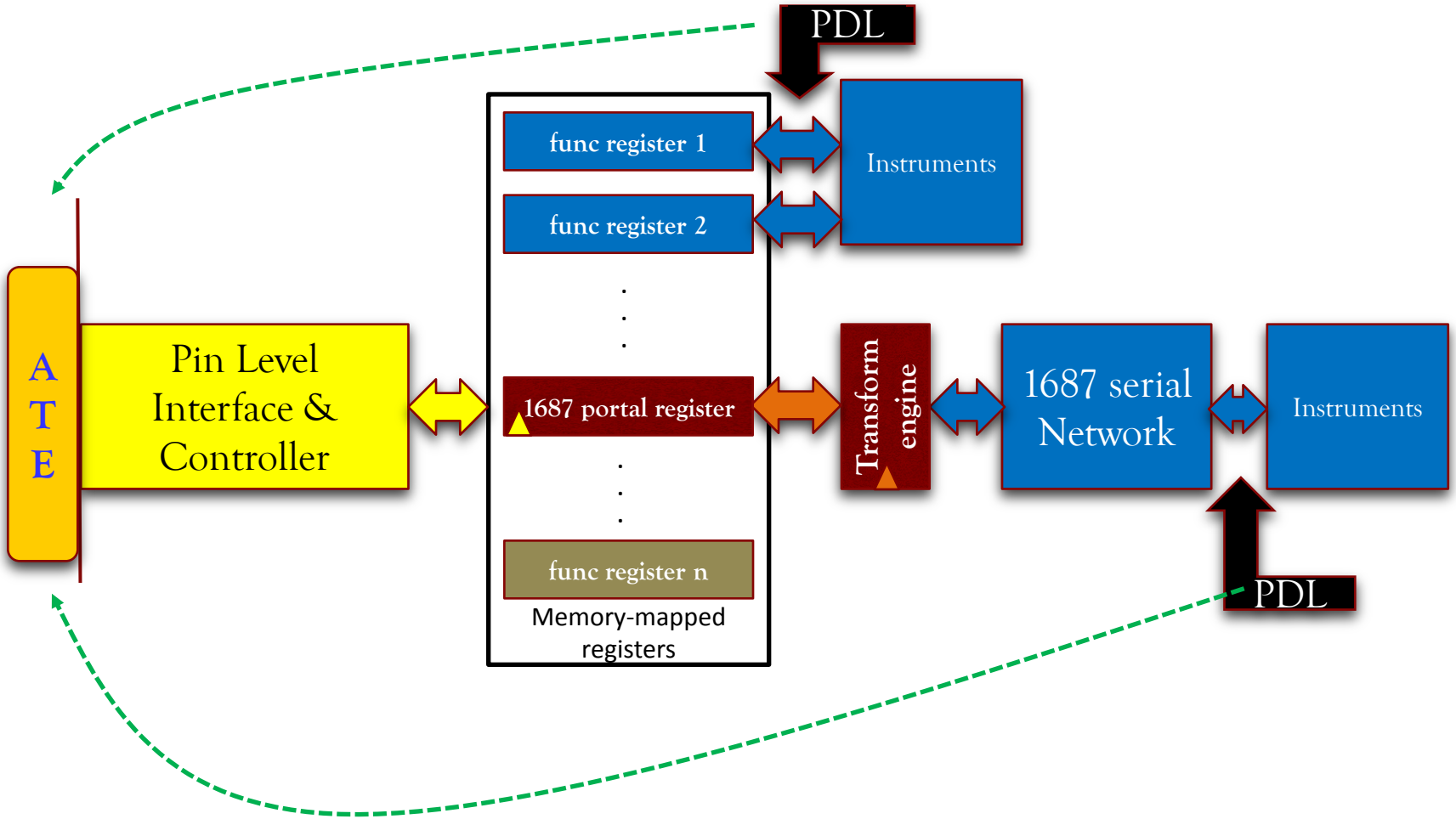
Pressing issues

- Where do we draw the line for the callback endpoints?
- Does P1687.1 need to standardize any hardware (e.g. the Transformation Engine) or can we simply do everything in software? (transactions & callbacks, with iApply as the defining event in a transaction)
  - Proposal: let's do both: show the hardware as an example implementation, and show a callback which incorporates it, but allow other hardware implementations and only require that the software callback be delivered. We should list the constraints for interoperability that must be met (like the figure in 1687 showing the flow through CSU).
- Relationship of P1687.1 to SJTAG
  - Does P1687.1 stop at the device boundary?
  - Can the callback idea be common between the two standards?
  - If so, to what degree do we need to align? Language? Style? Scope? API?

From 6/27/2017 meeting

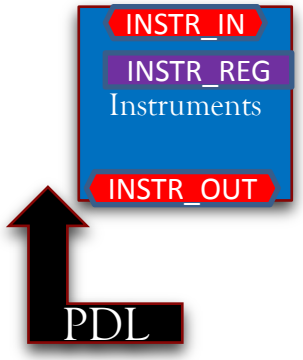
PDL RETARGETING EXAMPLE

Example for use model 2: “callback” idea (0)



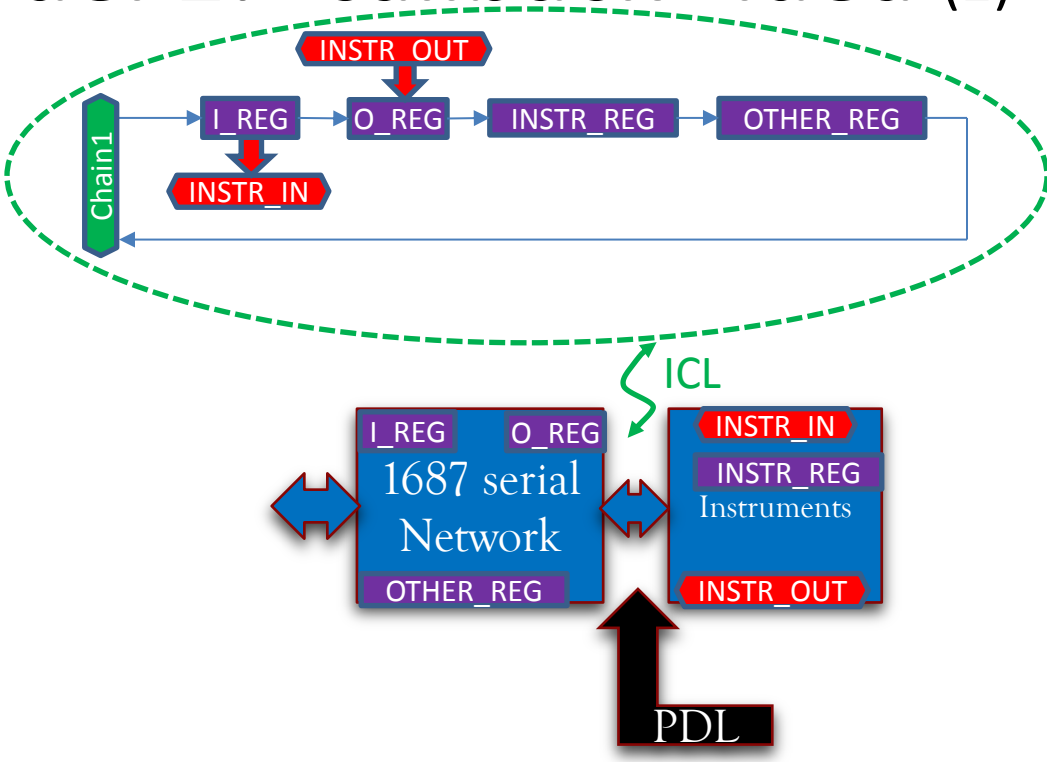
End goal: retarget PDL to ATE pin events

Example for use model 2: “callback” idea (1)




```
iProc my_test {} {
  iWrite INSTR_IN 0b0101
  iWrite INSTR_REG 0xABCD
  iApply
  iRead INSTR_OUT 0b1111
  iApply
}
```

Example for use model 2: “callback” idea (2)




```
iScan Chain1 40 -si 0x50ABCD0000 \
                -so 0xXXXXXXXXXX
iApply
iScan Chain1 40 -si 0x0000000000 \
                -so 0xXFXXXXXXXX
iApply
```

```
iProc my_test {} {
  iWrite INSTR_IN 0b0101
  iWrite INSTR_REG 0xABCD
  iApply
  iRead INSTR_OUT 0b1111
  iApply
}
```

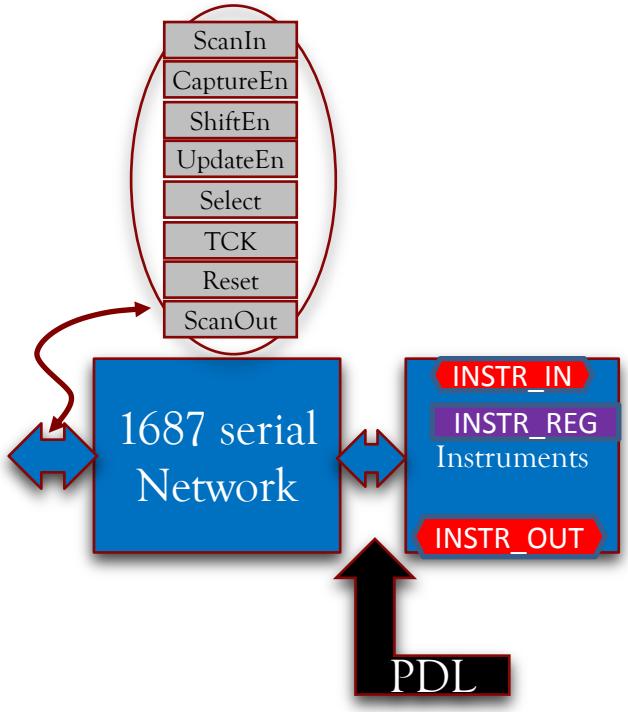
Retarget to ScanInteface “Chain1” (traditional 1687)

Example for use model 2: “callback” idea (3)

Same content, different format

```
// testbench actions
initial begin
   TCK = 0;
   Reset = 0;
   Select = 0;
   CaptureEn = 1;
   ShiftEn = 0;
   UpdateEn = 0;
   #1;
   TCK = 1;     // capture
   #1;
   TCK = 0;
   CaptureEn = 0;
   ShiftEn = 1;
   ScanIn = 0; // Chain1[0]
   #1;
   TCK = 1;     // shift
   #1;
   TCK = 0;
   ScanIn = 0; // Chain1[1]
   #1;
   TCK = 1;     // shift
  ...

end
```



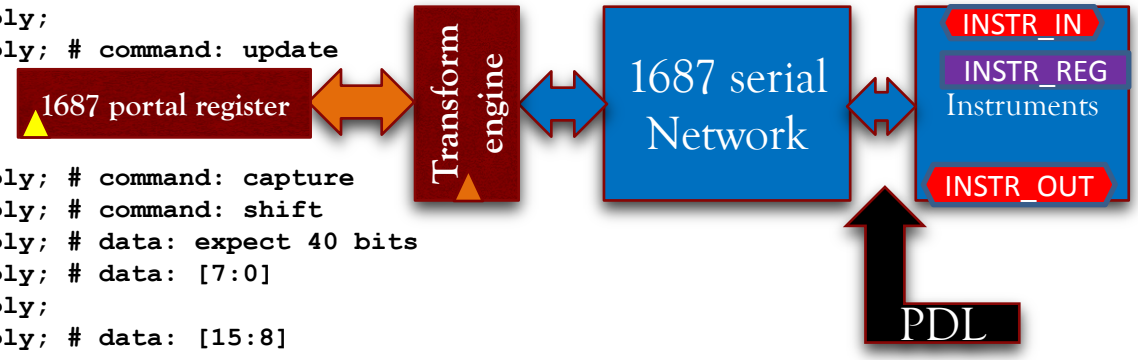

```
iProc my_test {} {
  iWrite INSTR_IN 0b0101
  iWrite INSTR_REG 0xABCD
  iApply
  iRead INSTR_OUT 0b1111
  iApply
}
```

Retarget as a Verilog TestBench

Example for use model 2: “callback” idea (4)

```
iWrite Portal1687 0b10000001; iApply; # command: capture
iWrite Portal1687 0b01000001; iApply; # command: shift
iWrite Portal1687 0b00101000; iApply; # data: expect 40 bits
iWrite Portal1687 0b00000000; iApply; # data: [7:0]
 iRead Portal1687 0bxxxxxxxx; iApply;
iWrite Portal1687 0b00000000; iApply; # data: [15:8]
 iRead Portal1687 0bxxxxxxxx; iApply;
iWrite Portal1687 0b11001101; iApply; # data: [23:16]
 iRead Portal1687 0bxxxxxxxx; iApply;
iWrite Portal1687 0b10101011; iApply; # data: [31:24]
 iRead Portal1687 0bxxxxxxxx; iApply;
iWrite Portal1687 0b01010000; iApply; # data: [39:32]
 iRead Portal1687 0bxxxxxxxx; iApply;
iWrite Portal1687 0b00100001; iApply; # command: update

# second iApply group (iRead)
iWrite Portal1687 0b10000001; iApply; # command: capture
iWrite Portal1687 0b01000001; iApply; # command: shift
iWrite Portal1687 0b00101000; iApply; # data: expect 40 bits
iWrite Portal1687 0b00000000; iApply; # data: [7:0]
 iRead Portal1687 0bxxxxxxxx; iApply;
iWrite Portal1687 0b00000000; iApply; # data: [15:8]
 iRead Portal1687 0bxxxxxxxx; iApply;
iWrite Portal1687 0b00000000; iApply; # data: [23:16]
 iRead Portal1687 0bxxxxxxxx; iApply;
iWrite Portal1687 0b00000000; iApply; # data: [31:24]
 iRead Portal1687 0bxxxxxxxx; iApply;
iWrite Portal1687 0b00000000; iApply; # data: [39:32]
 iRead Portal1687 0bxxxx1111; iApply;
 iNote -status "Compare INSTR_OUT 0b111";
iWrite Portal1687 0b00100001; iApply; # command: update
```




```
iProc my_test {} {
  iWrite INSTR_IN 0b0101
  iWrite INSTR_REG 0xABCD
  iApply
  iRead INSTR_OUT 0b1111
  iApply
}
```

```
0x50ABCD0000   0xXXXXXXXXXX
0x0000000000   0xXFXXXXXXXX
```

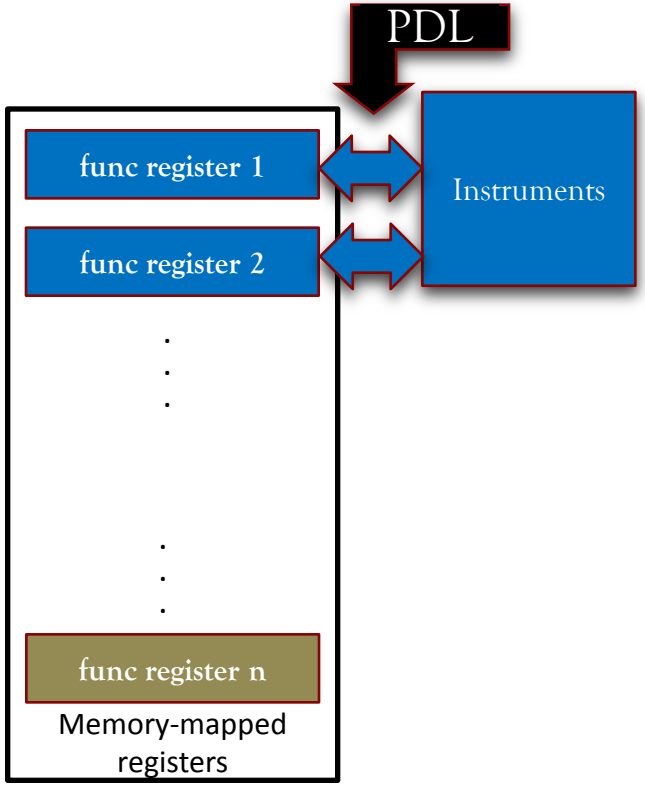
**to do list: we need to formalize the iNote pragma to track to PDL actions (colored bits) into the tester language*

Retarget to Portal register

[Using TransformEngine spec we’ll invent]

Example for use model 2: “callback” idea (5)

```
# actions through func regs
iWrite FuncReg1 0b00000001
iWrite FuncReg2 0b00000010
iApply
iWrite FuncRegi 0b01101001
iApply
iRead FuncRegn 0bxxxxxx01
iApply
...
```

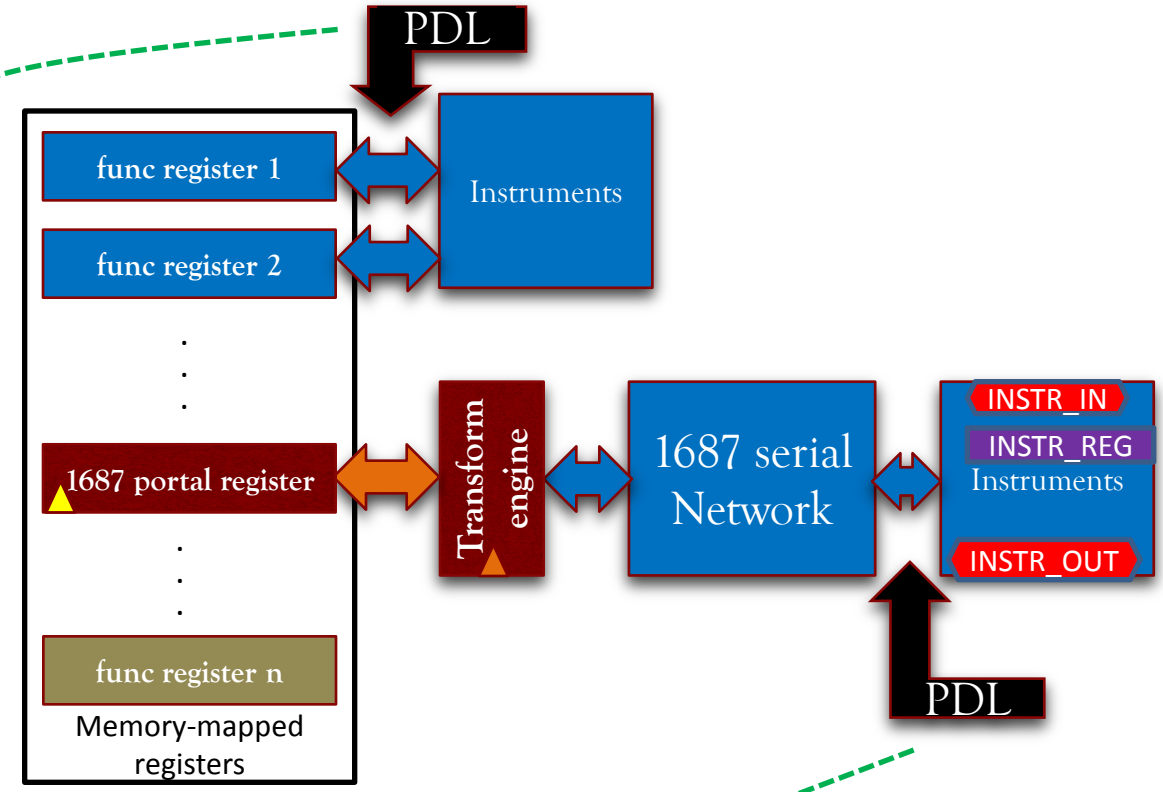


Functional tests accessing instruments

Example for use model 2: “callback” idea (6)

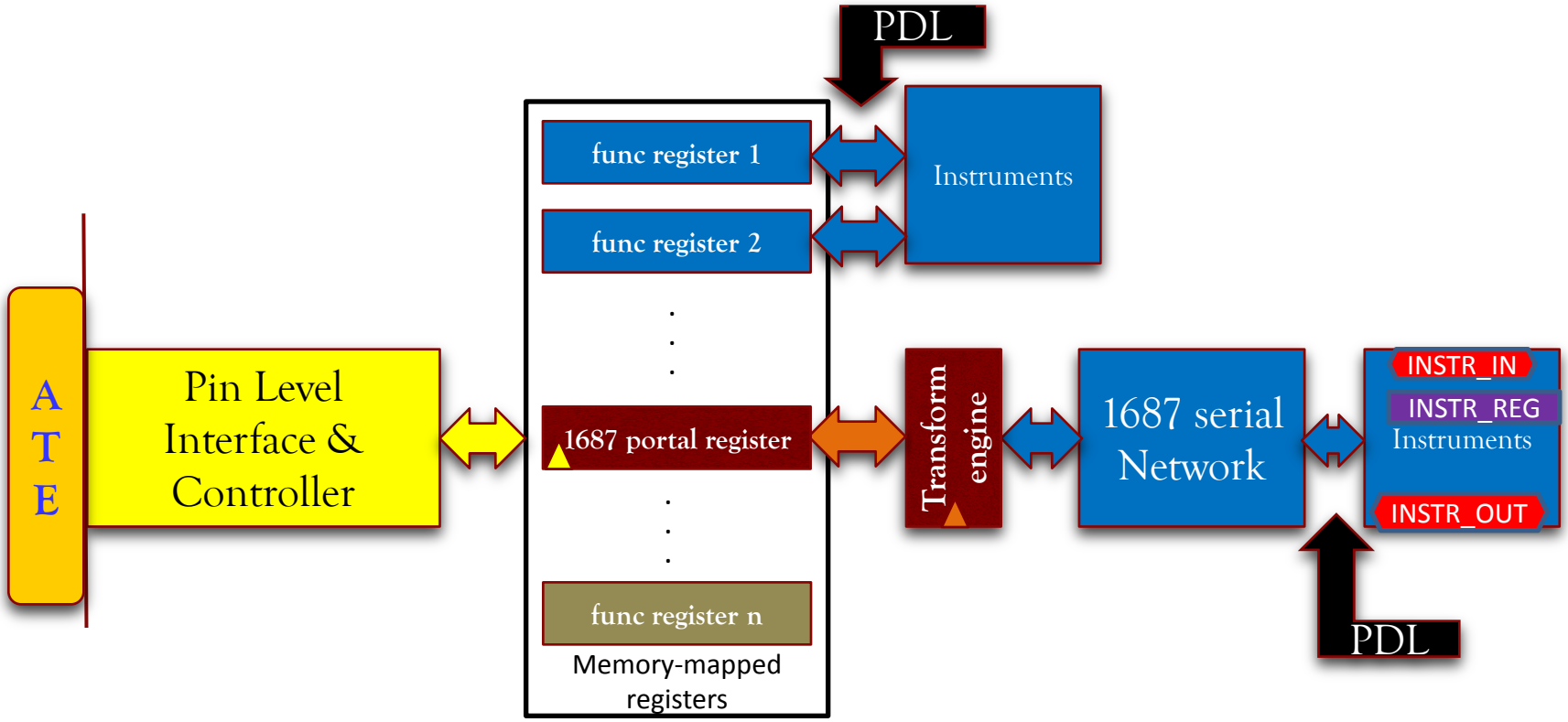
```
# actions through func regs
iWrite FuncReg1 0b00000001
iWrite FuncReg2 0b00000010
iApply
iWrite FuncRegi 0b01101001
iApply
iRead FuncRegn 0bxxxxxx01
iApply
...

# actions through 1687 Portal
iWrite Portal1687 0b10000001
iApply
iWrite Portal1687 0b01000001
iApply
iWrite Portal1687 0b00101000
iApply
iWrite Portal1687 0b00000000
iApply
iRead Portal1687 0bxxxxxxxx
iApply
iWrite Portal1687 0b00000000
iApply
iRead Portal1687 0bxxxxxxxx
iApply
iWrite Portal1687 0b11001101; iApply
iRead Portal1687 0bxxxxxxxx; iApply
iWrite Portal1687 0b10101011; iApply
iRead Portal1687 0bxxxxxxxx; iApply
iWrite Portal1687 0b01010000; iApply
iRead Portal1687 0bxxxxxxxx; iApply
iWrite Portal1687 0b00100001; iApply
...
```



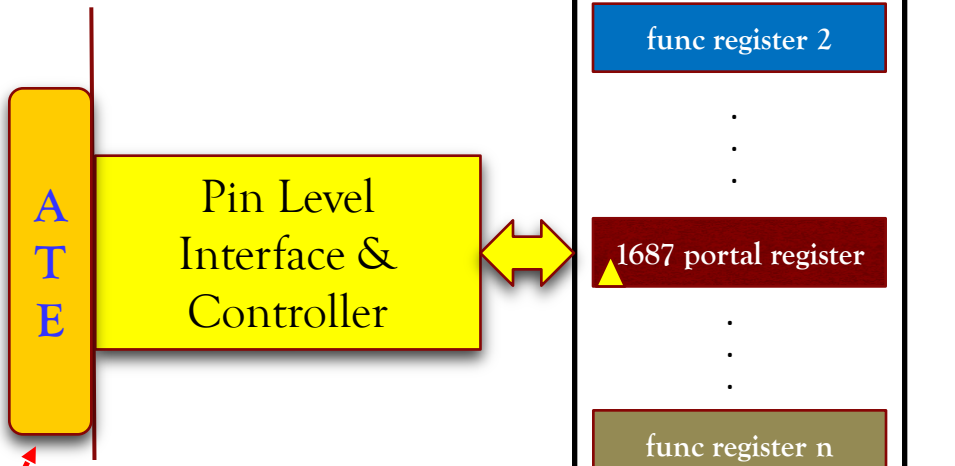
Almost there: all PDL has been retargeted to memory-mapped registers: one step left

Example for use model 2: “callback” idea (7)



Final step: how do we get from register writes and reads back to ATE pin events?

Example for use model 2: "callback" idea (8)



Define every data register:

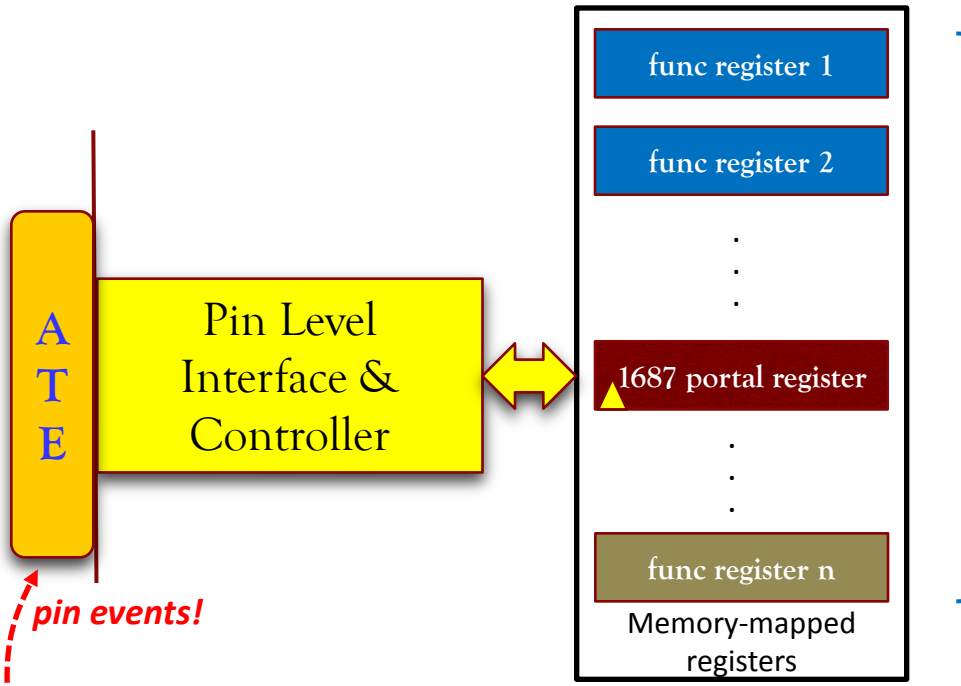
```
DataRegister register_name {
  //Common for all types
  Attribute att_name = att_value;
  ResetValue reset_value;
  DefaultLoadValue load_value;
  RefEnum enum_name;
  //Selectable
  WriteDataSource data_source;
  WriteEnSource enable_source;
  //Addressable
  AddressValue addr_value;
  //write callback
  WriteCallBack write_function_call;
  //read callback
  ReadCallBack read_function_call;
  ReadDataSource data_source;
}
```

pin events!

```
# bit-bang the I2C interface to deliver the write data
iProc Write_I2C_1687_Portal_register {write_data} {
   iCall SelectPortalAddress;
   iWrite SCL 0b0
   iWrite SDA $write_data[0]
   iApply
   iWrite SCL 0b1
   iApply
   iWrite SCL 0b0
   iWrite SDA $write_data[1]
   iApply
   ...
}
```

1687.1 optimization: hybridize the Addressable and Callback DataRegisters. Also, need to allow for callbacks to different interfaces if there are more than one way to talk to a register.

Example for use model 2: “callback” idea (8.5)



Define every data register:

```
DataRegister register_name {
  //Common for all types
  Attribute att_name = att_value;
  ResetValue reset_value;
  DefaultLoadValue load_value;
  RefEnum enum_name;
  //Selectable
  WriteDataSource data_source;
  WriteEnSource enable_source;
  //Addressable
  AddressValue addr_value;
  //write callback
  WriteCallBack write_function_call;
  //read callback
  ReadCallBack read_function_call;
  ReadDataSource data_source;
}
```

```
# bit-bang the I2C interface to deliver the write data
iProc Write_I2C_1687_Portal_register {write_data} {
  iCall SelectPortalAddress;
  iWrite SCL 0b0
  iWrite SDA $write_data[0]
  iNote -payload $write_data[0] # I_REG[3] == iScan Chain1[39] (from slide 8)
  iApply
  iWrite SCL 0b1
  iApply
  iWrite SCL 0b0
  iWrite SDA $write_data[1]
  iNote -payload $write_data[1] # I_REG[2] == iScan Chain1[38]
  iApply ... }
```

Proposed PDL0 iNote command option:
iNote –payload <ICL_reg_bit_name>
 # to identify PDL source bit for callback

Example C code for I2C write callback

```
int write_1687_portal_from_I2C(write_data)
char write_data[];
{
  int i = 0;
  int LOCAL_DEBUG = 0;

  if (LOCAL_DEBUG)
    printf("DEBUG: called write_1687_portal_from_I2C >%s<\n",write_data);

  if (strlen(write_data) == PORTAL_WIDTH) {
//  printf("iProc Write_I2C_1687_Portal_register {%s} {\n",write_data);
    printf("iCall SelectPortalAddress;\n");
    while (i < PORTAL_WIDTH) {
      printf("iWrite SCL 0b0\n");
      printf("iWrite SDA 0b%c\n",write_data[i]);
      printf("iApply\n");
      printf("iWrite SCL 0b1\n");
      printf("iApply\n");
      i++;
    }
//  printf("}\n");
    return(1);
  }
  else {
    fprintf(stderr,"ERROR: width of write_data (%d) does not match expected
(%d)\n",strlen(write_data),PORTAL_WIDTH);
    return(0);
  }
}
```

Note: this code lacks payload tracking!

```
iCall SelectPortalAddress;
iWrite SCL 0b0
iWrite SDA 0b0
iApply
iWrite SCL 0b1
iApply
iWrite SCL 0b0
iWrite SDA 0b0
iApply
iWrite SCL 0b1
iApply
iWrite SCL 0b0
iWrite SDA 0b1
iApply
iWrite SCL 0b1
iApply
iWrite SCL 0b0
iWrite SDA 0b0
iApply
iWrite SCL 0b1
iApply
iWrite SCL 0b0
iWrite SDA 0b0
iApply
iWrite SCL 0b1
iApply
iWrite SCL 0b0
iWrite SDA 0b0
iApply
iWrite SCL 0b1
iApply
iWrite SCL 0b0
iWrite SDA 0b0
iApply
iWrite SCL 0b1
iApply
iWrite SCL 0b0
iWrite SDA 0b1
iApply
iWrite SCL 0b1
iApply
```

Michele's example SIT (Simplified ICL Tree) file

```
TOP_Level
{
 INSTANCE Inst_U1 OF U1
 INSTANCE Inst_U5 OF U5
}

U1
{
 ACCESSINTERFACE Top_TAP JTAG 6 2
 {
  REGISTER U1_IR 6 Bypass: "0x3F"
  U1_subystem
  {
   REGISTER U1_Register 8 Bypass: "0x00"
   AI_TRANSLATOR U2_to_U1 JTAG_to_PDL U1_Register Inst_U2.U2_TAP Inst_U3.U3_TAP
    {
    INSTANCE Inst_U2 OF U2
    {
     INSTANCE Inst_U4 OF U4
    }
    INSTANCE Inst_U3 OF U3
    }
  }
 }
}

U2
{
 ACCESSINTERFACE U2_TAP JTAG 4 2
 {
   REGISTER U2_IR  4   Bypass: "0x3F"
   REGISTER U2_I2C 8   Bypass: "0x00"
   REGISTER U2_SPI 16  Bypass: "0x0000"
   AI_TRANSLATOR U4_to_U2 I2C_to_PDL U2_I2C Inst_U4.U4_I2C
   AI_TRANSLATOR U4_to_U2 SPI_to_PDL U2_SPI Inst_U4.U4_SPI
 }
}

U3
{
 ACCESSINTERFACE U3_TAP JTAG 4 2
 {
   REGISTER U3_IR         4   Bypass: "0x3F"
   REGISTER U3_Register   12  Bypass: "0x000"
 }
}

U4
{
 ACCESS_INTERFACE U4_SPI SPI "0x90 ,0x80"
  {
   REGISTER U4_SPI_Reg 8   Bypass: "0x00"
  }
 ACCESS_INTERFACE U4_I2C I2C "0x10"
  {
   REGISTER U4_I2C_Reg 10   Bypass: "0x000"
  }

 }


U5
{
 AI_TRANSLATOR U67_to_U5 I2C_to_USB U5_I2C_Master
 ACCESS_INTERFACE U5_I2C_Master I2C "0x30, 0x40"
  {
    INSTANCE U6 OF Sub_module
    INSTANCE U7 OF Sub_module
  }
}

Sub_module
 {
   REGISTER this_register 10   Bypass: "0x000"
 }
```