

I2C protocol

ASD contest 2011

ASD user event, October 4th 2011

Terry Dennemans

ASD User Event Presentation
October 4, 2011

verum[®]

Contents

- Introduction
- Remote I/O home project
- Architecture / Decomposition
- I2C HW protocol
- ASD models from HW to UI
- Problems and solutions
- Conclusions / lessons learned

Introduction

- For the contest, my goal was:
 - Show different layers of SW where ASD can be used
 - Show how to deal with data using foreign components
 - Make big model(s) in short time → no optimizations of state models
 - Get a working example (extend if time allows)
 - Integrate ASD in c#
 - Start the SW part of my home project

Remote I/O home project

- Generic I/O board with several kind of input/output signals
- I/O boards will be used for home automation (control lights, pumps, valves, detect switches)
- I/O should be accessible over Ethernet home network
- Programmable I/O module (over ethernet)

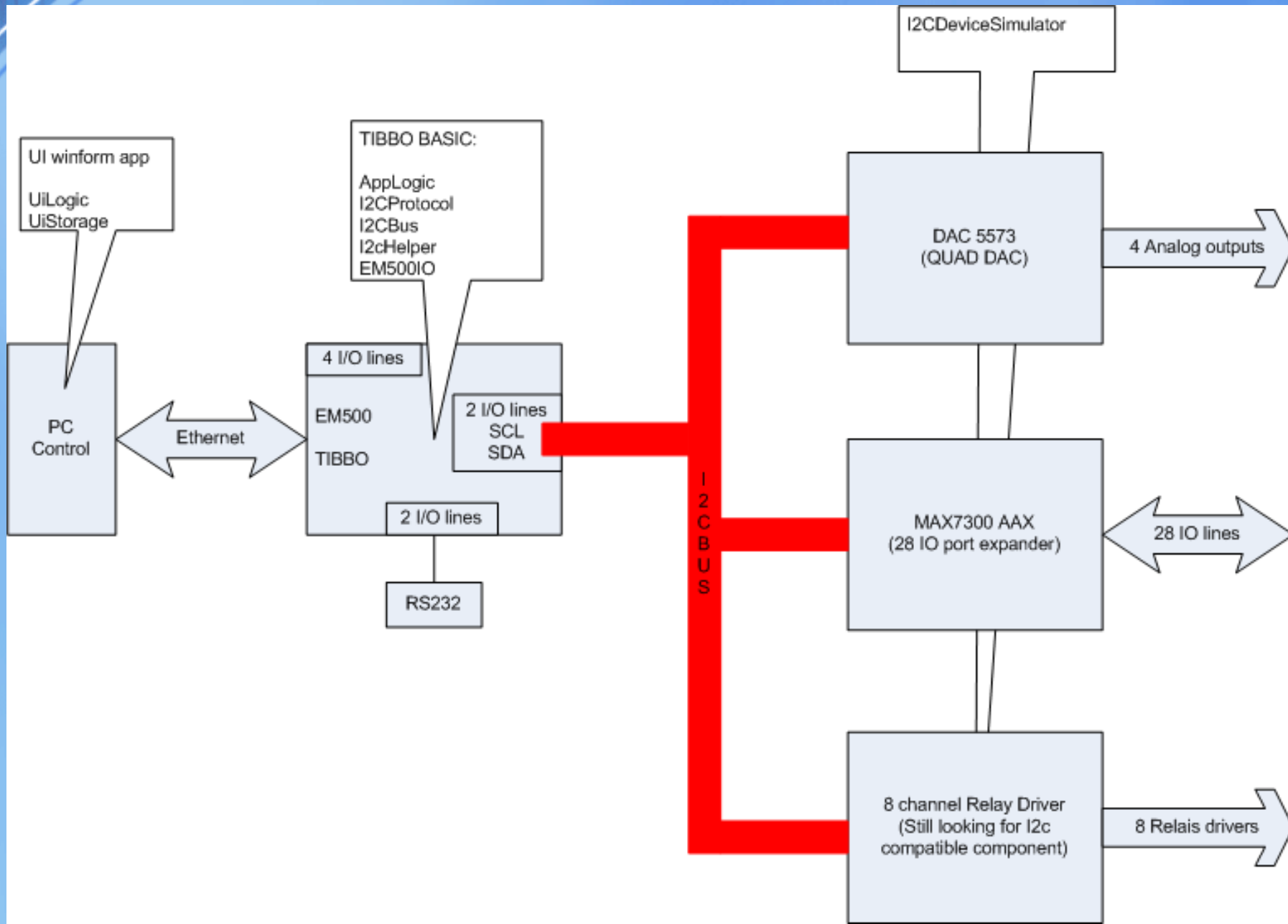
→ Tibbo EM500 device was chosen:

- BASIC-programmable Ethernet Module with 8 I/O lines
- 512KBytes of flash and 208 bytes of EEPROM memory
- Tcp/ip socket support

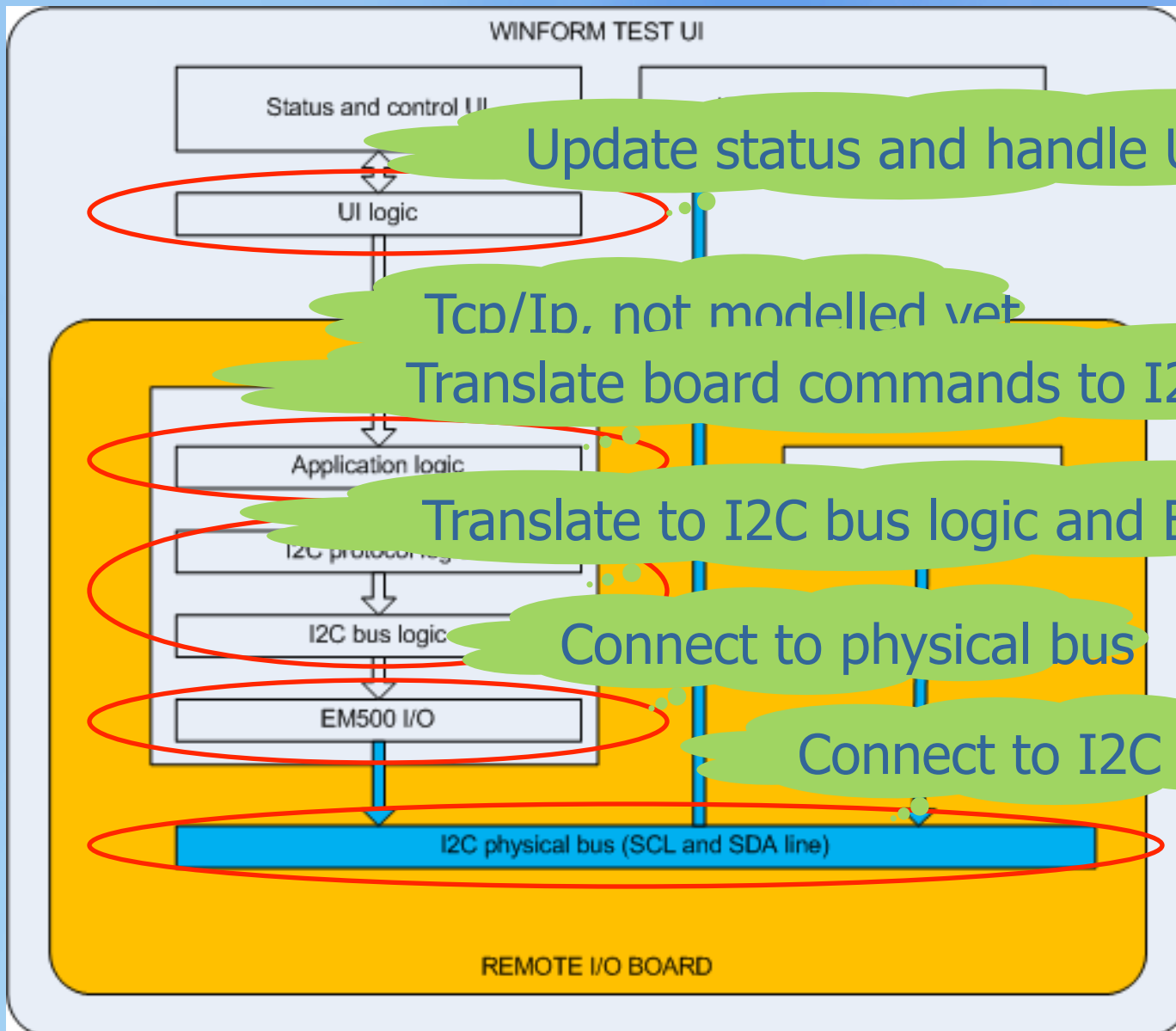
- Limited # I/O lines → use I2C HW protocol to control devices like
 - I/O expander
 - DACs
 - Relay drivers



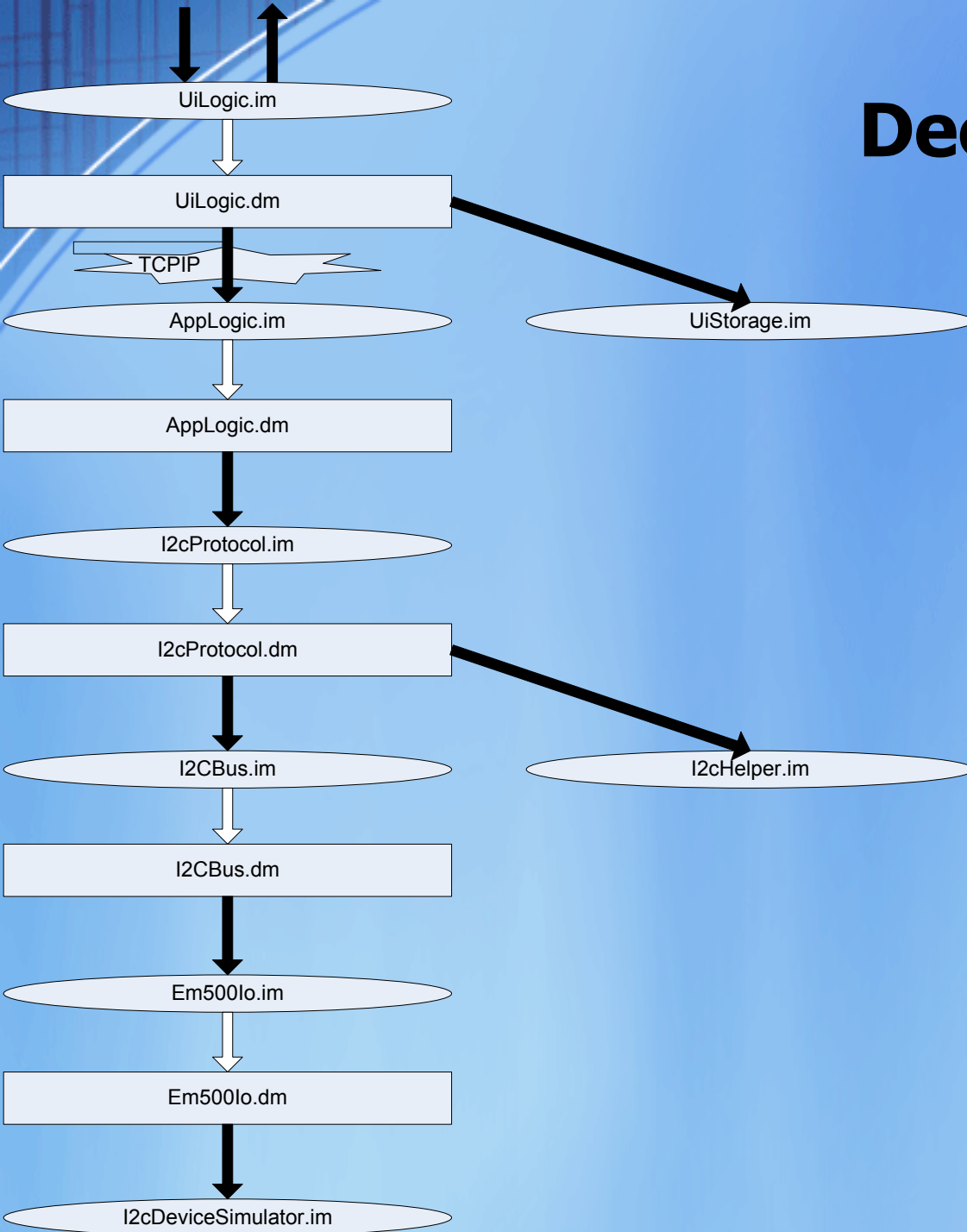
System HW Architecture



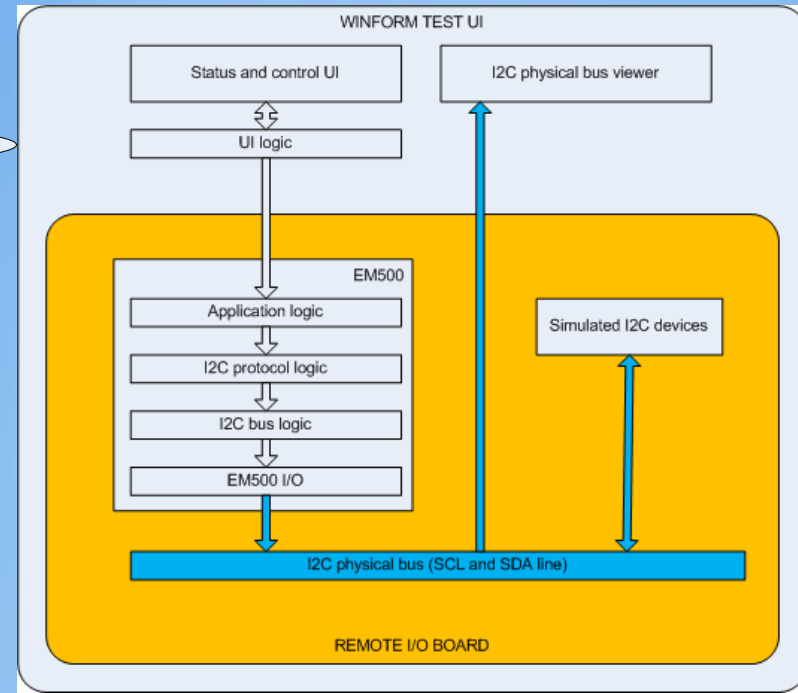
SW Architecture: ASD at different SW layers



Decomposition



- [UiLogic](#)
- [UIStorage](#)
- [AppLogic](#)
- [I2CProtocol](#)
- [I2cHelper](#)
- [I2CBus](#)
- [Em500IO](#)
- [I2cDeviceSimulator](#)



The resulting application

EM500 remote IO

MAX7300 AAX 28 ports I/O (I2C)

| Configuration (I/O) | State |
|--|---|
| I/O 00 <input checked="" type="checkbox"/> Input | <input checked="" type="checkbox"/> Active High |
| I/O 01 <input checked="" type="checkbox"/> Input | <input type="checkbox"/> Active High |
| I/O 02 <input checked="" type="checkbox"/> Input | <input checked="" type="checkbox"/> Active High |
| I/O 03 <input checked="" type="checkbox"/> Input | <input type="checkbox"/> Active High |
| I/O 04 <input checked="" type="checkbox"/> Input | <input checked="" type="checkbox"/> Active High |
| I/O 05 <input checked="" type="checkbox"/> Input | <input type="checkbox"/> Active High |
| I/O 06 <input checked="" type="checkbox"/> Input | <input checked="" type="checkbox"/> Active High |
| I/O 07 <input checked="" type="checkbox"/> Input | <input type="checkbox"/> Active High |
| I/O 08 <input checked="" type="checkbox"/> Input | <input checked="" type="checkbox"/> Active High |
| I/O 09 <input checked="" type="checkbox"/> Input | <input checked="" type="checkbox"/> Active High |
| I/O 10 <input checked="" type="checkbox"/> Input | <input type="checkbox"/> Active High |
| I/O 11 <input checked="" type="checkbox"/> Input | <input type="checkbox"/> Active High |
| I/O 12 <input checked="" type="checkbox"/> Input | <input checked="" type="checkbox"/> Active High |
| I/O 13 <input checked="" type="checkbox"/> Input | <input checked="" type="checkbox"/> Active High |
| I/O 14 <input checked="" type="checkbox"/> Input | <input type="checkbox"/> Active High |
| I/O 15 <input checked="" type="checkbox"/> Input | <input type="checkbox"/> Active High |
| I/O 16 <input type="checkbox"/> Input | <input type="checkbox"/> Active High |
| I/O 17 <input type="checkbox"/> Input | <input type="checkbox"/> Active High |
| I/O 18 <input type="checkbox"/> Input | <input type="checkbox"/> Active High |
| I/O 19 <input type="checkbox"/> Input | <input type="checkbox"/> Active High |
| I/O 20 <input type="checkbox"/> Input | <input type="checkbox"/> Active High |
| I/O 21 <input type="checkbox"/> Input | <input type="checkbox"/> Active High |
| I/O 22 <input type="checkbox"/> Input | <input type="checkbox"/> Active High |
| I/O 23 <input type="checkbox"/> Input | <input type="checkbox"/> Active High |
| I/O 24 <input type="checkbox"/> Input | <input type="checkbox"/> Active High |
| I/O 25 <input type="checkbox"/> Input | <input type="checkbox"/> Active High |
| I/O 26 <input type="checkbox"/> Input | <input type="checkbox"/> Active High |
| I/O 27 <input type="checkbox"/> Input | <input type="checkbox"/> Active High |

Get Configuration Get State
Set Configuration Set State

DACS (I2C)

| DAC 0 | DAC 1 | DAC 2 | DAC 3 |
|-------|-------|-------|-------|
| 32 | 64 | 128 | 255 |

Get State Set State

Clear log

RELAYS (I2C)

| | |
|---|---|
| Relay 0 <input checked="" type="checkbox"/> | Relay 4 <input checked="" type="checkbox"/> |
| Relay 1 <input type="checkbox"/> | Relay 5 <input type="checkbox"/> |
| Relay 2 <input checked="" type="checkbox"/> | Relay 6 <input checked="" type="checkbox"/> |
| Relay 3 <input type="checkbox"/> | Relay 7 <input type="checkbox"/> |

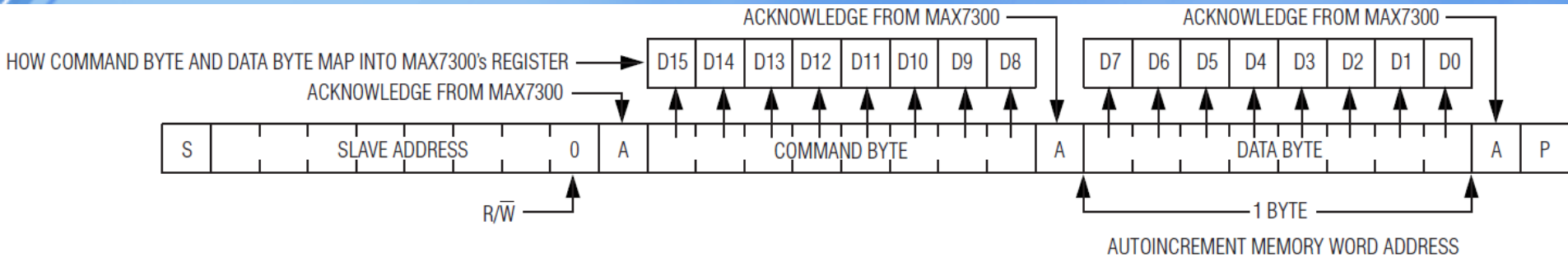
Get State Set State

I2C device simulator input/output logger

| SCL | SDA | Description |
|------|------|-------------------------|
| HIGH | HIGH | Initial IDLE state |
| HIGH | LOW | START I2C communication |
| LOW | LOW | Prepare for next bit |
| HIGH | LOW | Next bit, nr:7 |
| HIGH | HIGH | Received bit HIGH |
| LOW | HIGH | Prepare for next bit |
| HIGH | HIGH | Next bit, nr:6 |
| HIGH | LOW | Received bit LOW |
| LOW | LOW | Prepare for next bit |
| HIGH | LOW | Next bit, nr:5 |
| HIGH | LOW | Received bit LOW |
| LOW | LOW | Prepare for next bit |
| HIGH | LOW | Next bit, nr:4 |
| HIGH | LOW | Received bit LOW |
| LOW | LOW | Prepare for next bit |
| HIGH | LOW | Next bit, nr:3 |
| HIGH | LOW | Received bit LOW |
| LOW | LOW | Prepare for next bit |
| HIGH | LOW | Next bit, nr:2 |
| HIGH | LOW | Received bit LOW |
| LOW | LOW | Prepare for next bit |

Address received: 0x99 <Reading>
Set SDA low to acknowledge
ControlByte received: 0x12
Set SDA low to acknowledge
Set SDA low to acknowledge
STOP I2C communication
START I2C communication
Address received: 0x99 <Reading>
Set SDA low to acknowledge
ControlByte received: 0x14
Set SDA low to acknowledge
Set SDA low to acknowledge
STOP I2C communication
START I2C communication
Address received: 0x99 <Reading>
Set SDA low to acknowledge
ControlByte received: 0x16
Set SDA low to acknowledge
Set SDA low to acknowledge
STOP I2C communication

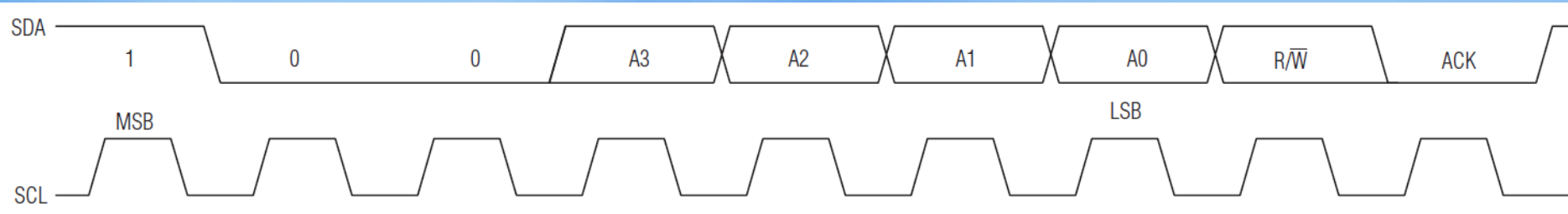
I2C protocol overview



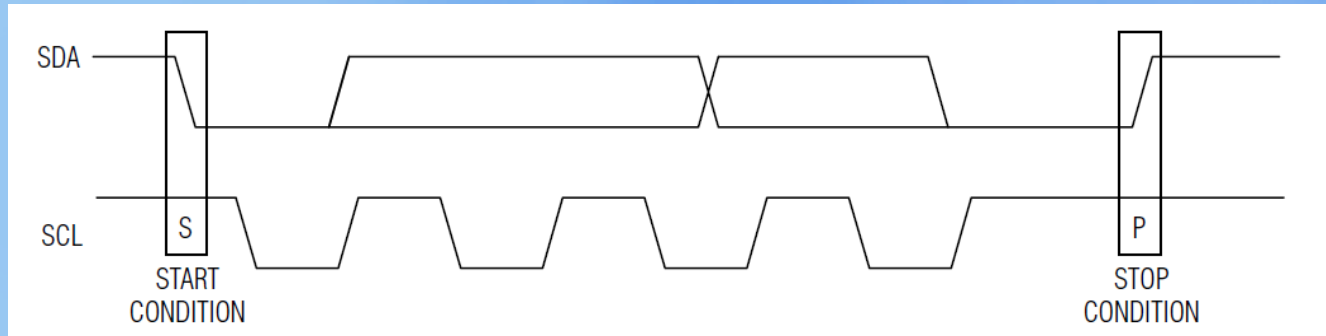
- Clock line and Data line (Serial communication)
- Message format:
 - First byte for addressing
 - Second byte for the control/command byte which defines what (register) to write/read.
 - The third (and following bytes) for the data.

I2C addressing

- The first byte of the message contains:
 - the 7-bit device address.
 - The 8th bit is to tell the device whether a read action (SDA is HIGH) or a write action (SDA is low) is chosen

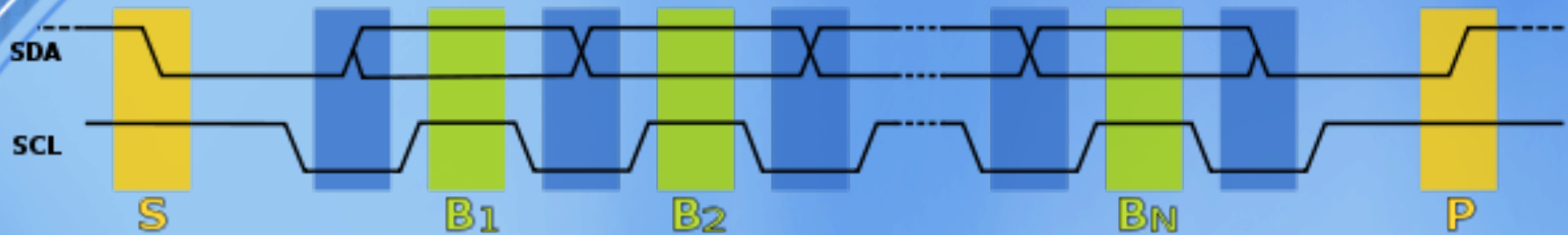


I2C Start / Stop condition



- IDLE: the SDA (data) and SCL (clock) lines are HIGH
- Message transfer start (S/START): the SDA line is pulled LOW
- End of message (P/STOP), the SDA line is pulled HIGH, making bus IDLE again.

I2C send/receive data

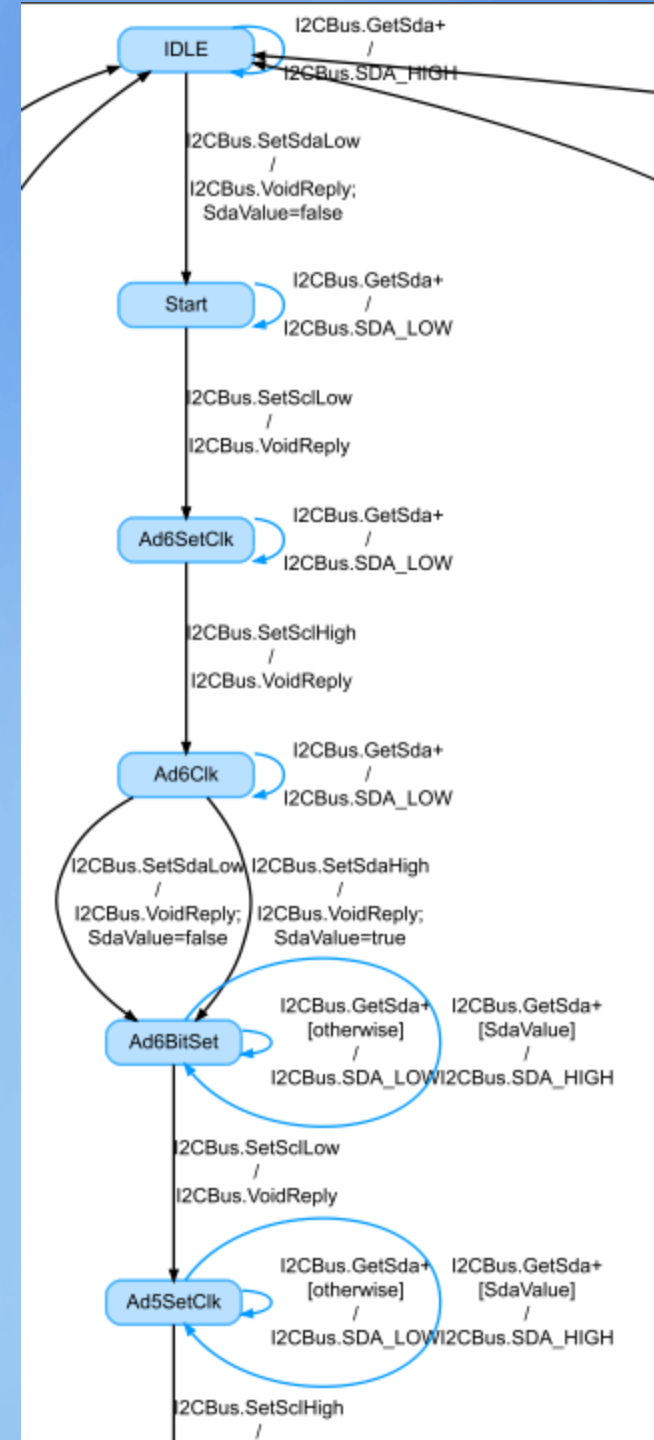


- Each data bit being send/received is clocked.
→ SCL line goes from LOW to HIGH.
When SCL is HIGH, SDA line represents the data bit.
- After having send 8 data bits, an acknowledge bit is being send by the receiver by pulling SDA line LOW.
- → So for each byte being send/received, 9 clock pulses will be generated (1 extra for the acknowledge).

I2C bus logic in ASD

- Interface model (102 states)
- Design model (204 states);
Each GetSDA requires 1 extra transition state

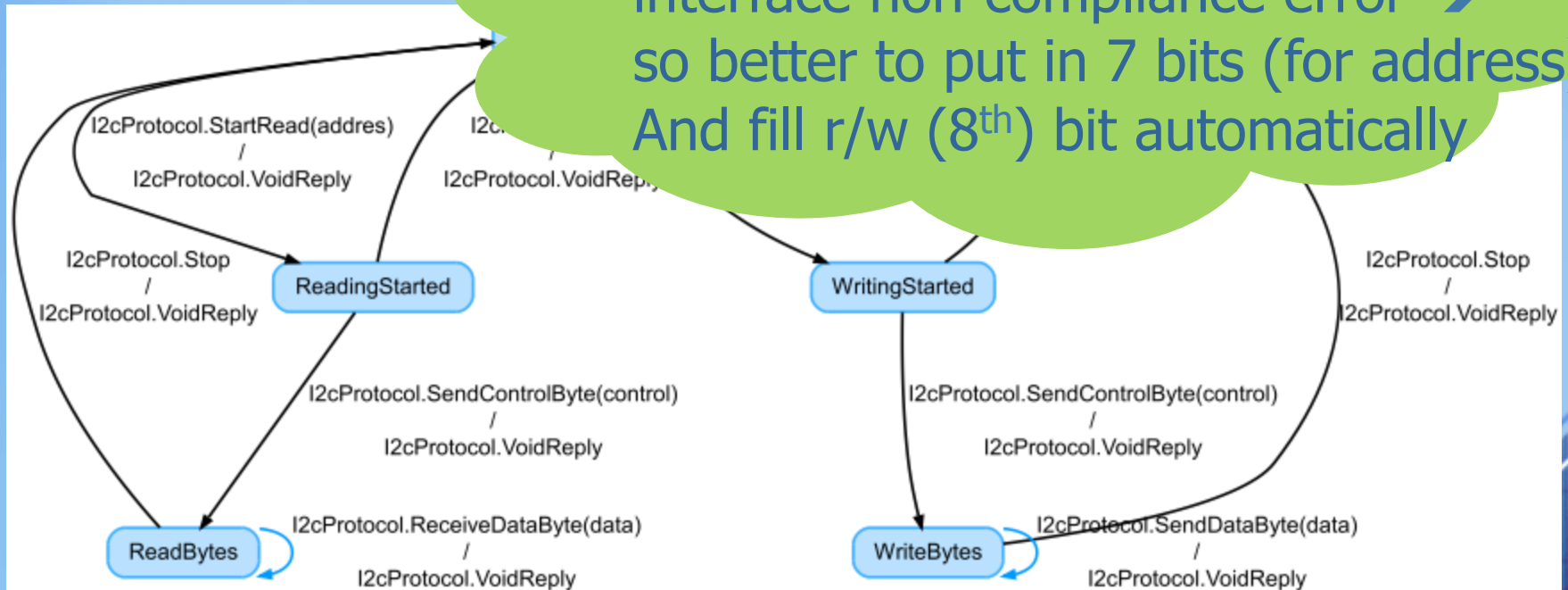
-> Show state model of
[I2CBus.im](#) or [I2CBus.dm](#)



I2C protocol in ASD

- StartRead([in]address:byte): void
- StartWrite([in]address:byte): void
- SendControlByte([in]control:byte): void
- SendDataByte([in]data:byte): void
- ReceiveDataByte(): byte
- Stop(): void

Model checker now chooses address that has wrong r/w flag resulting in interface non-compliance error → so better to put in 7 bits (for address) And fill r/w (8th) bit automatically



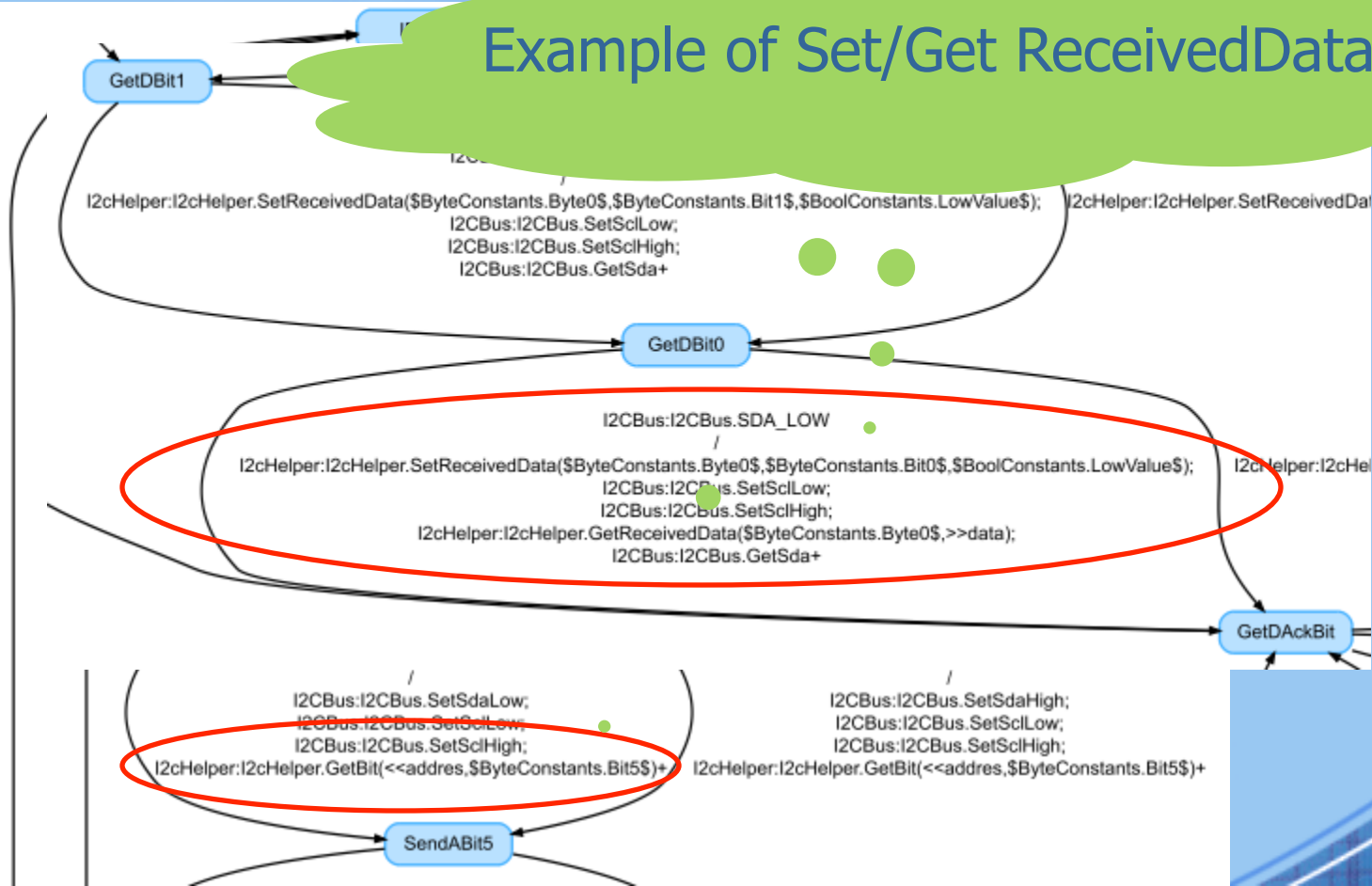
I2C protocol in ASD (DM)

- Use helper to deal with data



Example of GetBit(data, bitNr) helper
Returns Low or High value

Example of Set/Get ReceivedData



Application logic in ASD

| | Interface | Event | Guard | Actions | State Variable Updates | Target State |
|---|----------------------|--|-------|---|------------------------|--------------|
| 1 | IDLE <> | | | | | |
| 3 | AppLogic | SetIoConfig(cb0,cb1,cb2,cb3,cb4,cb5,cb6) | | I2cProtocol:I2cProtocol.StartWrite(\$ByteConstants.IoConfigWrite\$); I2cProtocol:I2cProtocol.SendControlByte(\$ByteConstants.IoConfigGroup0\$); I2cProtocol:I2cProtocol.SendDataByte(cb0); I2cProtocol:I2cProtocol.Stop; I2cProtocol:I2cProtocol.StartWrite(\$ByteConstants.IoConfigWrite\$); I2cProtocol:I2cProtocol.SendControlByte(\$ByteConstants.IoConfigGroup1\$); I2cProtocol:I2cProtocol.SendDataByte(cb1); I2cProtocol:I2cProtocol.Stop; I2cProtocol:I2cProtocol.StartWrite(\$ByteConstants.IoConfigWrite\$); I2cProtocol:I2cProtocol.SendControlByte(\$ByteConstants.IoConfigGroup2\$); I2cProtocol:I2cProtocol.SendDataByte(cb2); I2cProtocol:I2cProtocol.Stop; I2cProtocol:I2cProtocol.StartWrite(\$ByteConstants.IoConfigWrite\$); I2cProtocol:I2cProtocol.SendControlByte(\$ByteConstants.IoConfigGroup3\$); I2cProtocol:I2cProtocol.SendDataByte(cb3); | | IDLE |
| 4 | AppLogic | GetIoConfig(cb0,cb1,cb2,cb3,cb4,cb5,cb6) | | I2cProtocol:I2cProtocol.StartRead(\$ByteConstants.IoConfigRead\$); I2cProtocol:I2cProtocol.SendControlByte(\$ByteConstants.IoConfigGroup0\$); I2cProtocol:I2cProtocol.ReceiveDataByte(cb0); I2cProtocol:I2cProtocol.Stop; I2cProtocol:I2cProtocol.StartRead(\$ByteConstants.IoConfigRead\$); I2cProtocol:I2cProtocol.SendControlByte(\$ByteConstants.IoConfigGroup1\$); I2cProtocol:I2cProtocol.ReceiveDataByte(cb1); I2cProtocol:I2cProtocol.Stop; I2cProtocol:I2cProtocol.StartRead(\$ByteConstants.IoConfigRead\$); I2cProtocol:I2cProtocol.SendControlByte(\$ByteConstants.IoConfigGroup2\$); I2cProtocol:I2cProtocol.ReceiveDataByte(cb2); I2cProtocol:I2cProtocol.Stop; I2cProtocol:I2cProtocol.StartRead(\$ByteConstants.IoConfigRead\$); I2cProtocol:I2cProtocol.SendControlByte(\$ByteConstants.IoConfigGroup3\$); I2cProtocol:I2cProtocol.ReceiveDataByte(cb3); | | IDLE |
| 5 | AppLogic | SetIoState(port0,port1,port2,port3) | | I2cProtocol:I2cProtocol.StartWrite(\$ByteConstants.IoStateWrite\$); I2cProtocol:I2cProtocol.SendControlByte(\$ByteConstants.IoPort0\$); I2cProtocol:I2cProtocol.SendDataByte(port0); I2cProtocol:I2cProtocol.Stop; I2cProtocol:I2cProtocol.StartWrite(\$ByteConstants.IoStateWrite\$); I2cProtocol:I2cProtocol.SendControlByte(\$ByteConstants.IoPort1\$); I2cProtocol:I2cProtocol.SendDataByte(port1); I2cProtocol:I2cProtocol.Stop; I2cProtocol:I2cProtocol.StartWrite(\$ByteConstants.IoStateWrite\$); I2cProtocol:I2cProtocol.SendControlByte(\$ByteConstants.IoPort2\$); I2cProtocol:I2cProtocol.SendDataByte(port2); I2cProtocol:I2cProtocol.Stop; I2cProtocol:I2cProtocol.StartWrite(\$ByteConstants.IoStateWrite\$); I2cProtocol:I2cProtocol.SendControlByte(\$ByteConstants.IoPort3\$); I2cProtocol:I2cProtocol.SendDataByte(port3); | | IDLE |

ASD integration in C#

- Simple construction:

```
public partial class Form1 : Form, UIView, ISimulatorLogging
{ private UiLogicComponent uiLogicComponent;
  private UiLogic uiLogic;
```

```
public void StartAsd()
{
  uiLogicComponent = UiLogicComponent.GetInstance();
  uiLogicComponent.RegisterCB(this);
  uiLogicComponent.GetAPI(out uiLogic);
  uiLogic.GetIoConfig();
  uiLogic.GetIoState();
  uiLogic.GetRelayStates();
  uiLogic.GetDacStates();
}
```

- Foreign components:

- Generated stub
- Filled in implementation
- 1000 lines of code

- Form:

- 525 lines of code
- Connect to ASD UI logic
- Standard decouple UI updates using BeginInvoke

- Generated code:

- 12143 lines of code

Problems and solutions

- Deal with data → Use foreign components as helper components; you can do anything that you want
- Error in model, found when looking in detail at I2C physical output (*1st set Data, 2nd set Clk HIGH, now reversed*) →
 - Make sure you review your interface models
 - Make functional test at ASD boundaries to check requirements (*Dependency injection will be available soon, which makes testing using mocks easy*)
- Interface non compliance error was hard to find; *the ASD debugger showed incorrectly that the I2cBus model was in the IDLE state.* → new improved version of ASD tool offers even better debugging support and shows correct state

Conclusions / lessons learned (1)

- UI and application logic is not making use of full power of ASD:
 - Now only interface validations are done (*The actions done on the layers below*)
 - UI state is dealt with in foreign component (*because it is data intensive*), but in most cases you can define the UI states in ASD
 - It shows you can do almost anything using ASD !!
- The I2C bus logic models have become too big:
 - Positive is that they are still readable and state chart still looks fine
 - But it is a lot of work to fix the interface model issue (*reverse SDA set with CLK HIGH; Is there a find and replace??*)
 - → Need to optimize/refactor models; *e.g. by introducing sub state machines to send/receive bits and bytes*

Conclusions / lessons learned (2)

- Data restrictions in interface methods:
 - Try to avoid being able to put in invalid data (*e.g. certain combinations are never allowed, so best is that you can never choose these combinations*)
 - Or make your interface robust and give back an error in case of bad data input (*now interface model specified illegal*)
- Functional tests at ASD boundaries are still required:
 - Error free model check doesn't mean model does what you want/need (*e.g. the error in my models, or data is not taken into account*)
 - Refactoring of the models becomes easy if you have a safety net of tests at the ASD boundaries
 - Extend model check with external tests to cover all possible scenarios (*Define rules at the boundaries*)

Conclusions / lessons learned (3)

- Integration in C# was easy
- You see the ASD tool becoming better and better; debug support is improved a lot
- Home project status:
 - SW Interfaces are defined
 - Low-level test application is available
 - Write external tests
 - Refactor models and fix error
 - Generated code needs to be rewritten in Tibbo Basic
Or select different processor module supporting C
 - Develop remote I/O PCB (HW part)

Questions

??????