I^2C bus

deadline 30.06.2007

June 13, 2007

Introduction

The I^2C (Inter-Integrated Circuit) bus was designed as a simple bus to connect chips for application that do not need very high speeds. (The standard of 2000 mentions a standard version of 400kHz and a special version of 3.4MHz.) The bus uses just two wires, which means that all transfers happen in a serialized form. The idea is to first send the address is and then the data. It is a multi-master bus, meaning that several chips may try to write at the same time and it has a nice bus arbitration mechanism. When two chips are trying to write at the same time, they both keep writing until one of them notices that the signals on the bus do not agree with the signal it is trying to send and then wait for the other party to complete its operation before retrying.

Good descriptions of the I^2C bus can be found online, also a copy of the official 2.1 standard can be found on the course website.

What to model and verify.

Modeling the full standard would be too much work and lead to models which are too big. So we restrict to a system consisting of two masters M1 and M2 and two slaves S1 and S2. M1 must send a message to S1 and M2 must send a message to S2. You should verify that whatever happens, eventually S1 and S2 receive their messages.

To keep the task manageable and to keep the risk of state space explosion small:

- Forget about the R/W bit.
- Forget about the acknowledgment.

The aim is a working model that shows how the bus arbitration mechanism works rather than a full implementation of the standard.

Some suggestions

The file `wire.xml` provide an examples of how to model a wire. Pulling the wire low and releasing the wire is modeled as sending messages on a channel. As a result of these actions a boolean value, representing the state of the wire is changed. How this is done can be seen in Fig. 1.

In modeling a chip, you have to take care of two things: the protocol and the wire. It is possible to take care of both things in a single automaton, but it makes sense to split the functionality into two automata. That is a master chip splits into a transmitter and a sender and a slave chip splits into a listener and a receiver. This setup uses two instances of the wire automaton to control the booleans modeling the SCL and

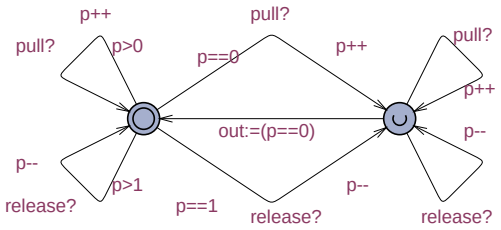

Figure 1: Uppaal model of a wire.

SDA wires. Their functions would be as follows:

listener	monitors the variables that model the SDA and SCL wires and passes start/stop and 0/1 events to the receiver by means of communication on channels.
receiver	Reacts to bus events and accepts a message iff the address matches.
transmitter	Accept instructions from the sender (start,stop,0,1), try to generate them on the bus using sending to the wire processes, monitor the variables that model the SDA and SCL wires to see if the operations succeeded and reply with ok or fail accordingly.
sender	Repeat trying to send a message until it succeeds.

Requirements

- The deadline is 30.06.2007, 23:59.
- The addresses and the message are supposed to be N-bit words, where N is a constant. You may test with $N = 2$, but it should be really easy to change it to any other value. That is, do not hardcode the length of the address into the automaton.

You will need to submit:

- The Uppaal model as .xml file.
- The Uppaal queries as a .q file.
- You can use the wire.xml example, you are *not* required to.
- You can use comments in declaration sections to explain important parts of your model.
- A **short** note describing your design decisions is optional.

Remember that you may be asked to explain your code/decisions.