Verification of Cyber-Physical System
Fall 2017

Exercice Sheet 8

Author : Sylvain Julmy

Professor : Ultes-Nitsche Ulrich
Assistant : Prisca Dotti

Sender

The idea to implement the sender, is to create a generic sender no matter how many bit we send. We just indicates the bit to send and the automaton will switch the signal value w.r.t. the current bit to send.

The figures 1 illustrate the times automaton design for the sender. Basically, the automaton wait in the central state for $\frac{\Delta^T}{2}$ and will move to the next state depending on the current signal value and the next bit to send.

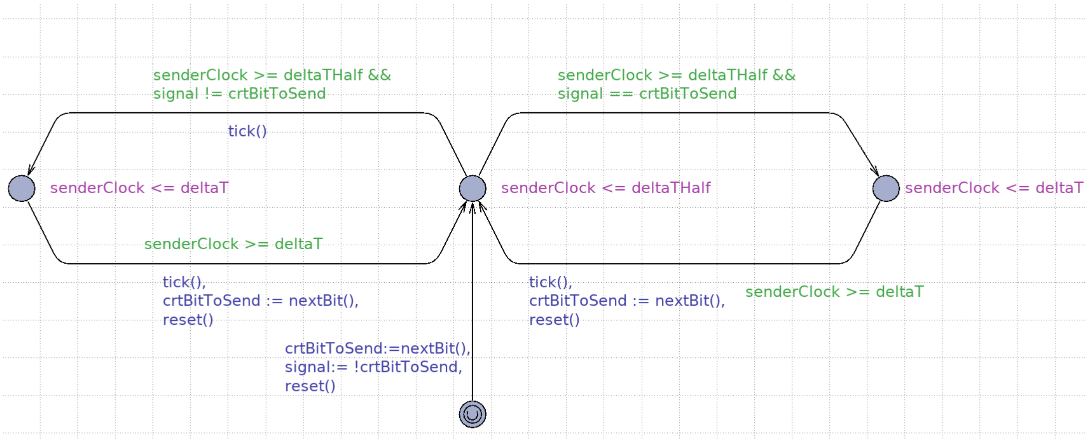

Figure 1: Sender automaton design

If the signal value is the same as the next bit to send, we have to tick before moving to either the left or right state.

Then both state do the same thing by ticking the signal, reseting the clock and computing the next bit to send.

The following listing is the implementation of the various function use for the sender automaton.

```
const int deltaT = 10000;
const int deltaTHalf = deltaT/2;
clock senderClock;
int bitCount=0;

bool preamble[64] = {
    1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,
    1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,
    1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,
    1,0,1,0,1,0,1,0,1,1};

bool data[64] = {
    1,1,1,0,0,0,1,0,1,1,1,0,1,0,1,1,0,0,
    1,0,1,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0,
    1,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,1,0,
    1,1,0,0,0,0,0,0,1,1};

bool crtBitToSend;

void reset()
{
    senderClock := 0;
}

// return the next bit to send
bool nextBit()
{
    int itr = bitCount % 64;
    if(bitCount > 127) return 1;
    if(bitCount++ < 64) {
        return preamble[itr];
    } else {
        return data[itr];
    }
}

void tick()
{
    signal := !signal;
}
```

Receiver

The receiver is not finish yet, but the idea is the following :

- Initialize Δ^t to a random value,not equals to 0
- During the preamble period, we adjust Δ^T with a specific learning rate (like 0.1)
- Detect the double 1 bit which indicates the end of the preamble
- Receive the data

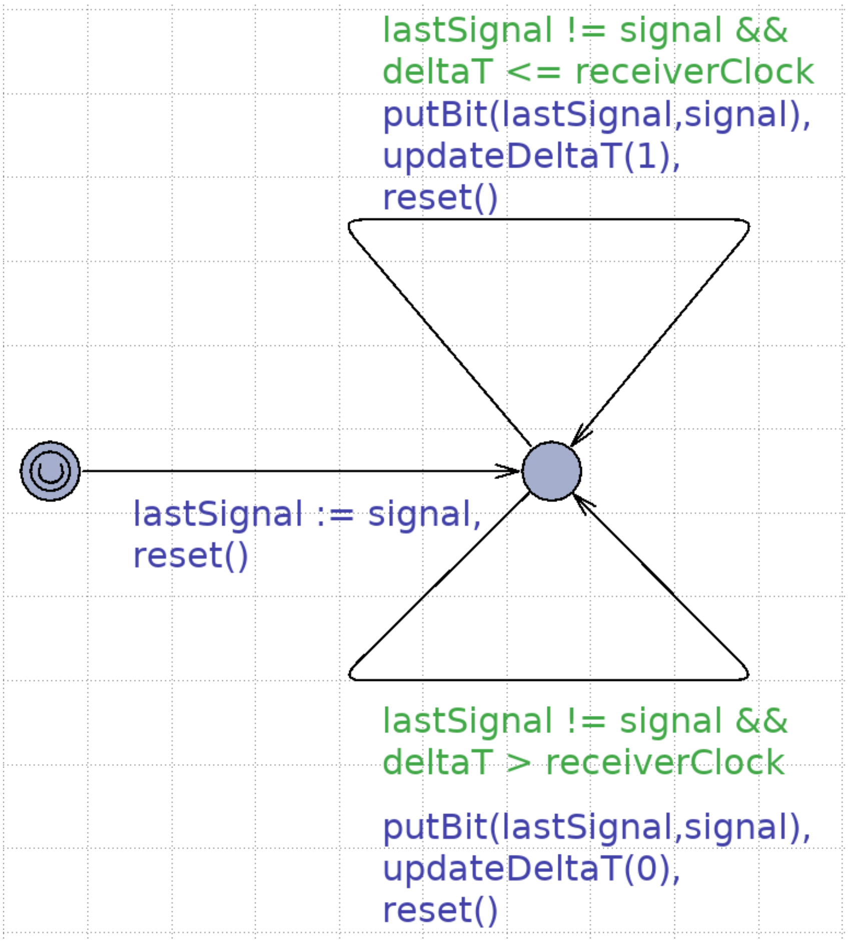

Figure 2: Receiver automaton design

The following listing is the implementation of the various function and variables use by the sender.

```
clock receiverClock;

// array to receive the data and compare at the end
bool data[64] = {
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0};

bool preamble[64] = {
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0};

int crtBitToReceiveIndex = 0;
double deltaT = 100.0;

// learning rate
double lr = 0.1;

bool lastSignal;

void reset()
{
    receiverClock := 0;
}

int bitCount = 0;

// PRE lastSignal != signal
void putBit(bool lastSignal,bool signal) {
    bool bit;
    int itr = bitCount % 64;

    if(lastSignal && !signal) {
        bit = 1;
    } else {
        bit = 0;
    }

    if(bitCount < 64) {
        preamble[itr] = bit;
    }
    else {
        data[itr] = bit;
    }
    bitCount++;
}

// update deltaT during the receive of the preamble
void updateDeltaT(bool direction) {
    if(direction) { // need to decrease deltaT
        deltaT := deltaT - deltaT * lr;
    }
    else { // otherwise deltaT is to small
        deltaT := deltaT + deltaT * lr;
    }
}
```