EECS 409
Problem 4.GP Results

Ahmad Al-Hammouri
E-mail: ata5@case.edu

1 Problem Statement

The problem I am proposing to work on is based on the Corbett’s distributed controller found in ”A User Guide to HyTech”. I want to extend the system such as the two sensors communicate with the controller through a communication channel. This channel introduces a constant overall delay of amount $\delta$ milliseconds. The goal is to determine the values of $\delta$ that result in acceptable time differences between controller signals. The time difference between controller signals was called $c$ in the HyTech guide.

2 The Channel Model

To simplify the problem, we assume there is no buffer at the channel’s input. Therefore, The incoming message is transmitted if it arrives when the channel is empty and it is dropped out if the channel is busy. This allows us to assume that the sum of messages’ transmission and propagation delays is constant provided that all messages are of equal size.

We will investigate two types of channels, full-duplex and half-duplex. In the first type, transmission in both directions can occur simultaneously whereas in the second, transmission can occur only in one direction at a given time. The models of the full-duplex and the half-duplex channels are shown in Figures 1 and 2, respectively.

In both figures, $Sensor_1$ uses the $data_1.Tx$ event to send the data through the channel, and the $ack_1.Rx$ event to receive the ack. The behavior of $Sensor_2$ is similar of $Sensor_1$’s except replacing 1 with 2. The controller, on the other hand, uses $data_1.Rx$, $data_2.Rx$, $ack_1.Tx$, and $ack_2.Tx$ to receive data from $Sensor_1$, to receive data from $Sensor_2$, to send ack to $Sensor_1$, and to send ack to $Sensor_2$, respectively. All of these events are synchronized with the channel’s similar events.

3 The Original System Changes

The original system automata must be changed as follows:

- $Sensor_1$ automata.
Figure 1: The Full duplex channel model. (left) transmitting in forward direction (the data from sensors to controller). (right) transmitting in backward direction (the ack from controller to sensors).

Figure 2: The Half duplex channel model.
Figure 3: The original system’s automata

- send₁ is replaced with data₁.Tx.
- ack₁ is replaced with ack₁.Rx.

• Sensor₁ automata.
  - send₂ is replaced with data₂.Tx.
  - ack₂ is replaced with ack₂.Rx.

• Controller automata.
  - send₁ is replaced with data₁.Rx.
  - ack₁ is replaced with ack₁.Tx.
  - send₂ is replaced with data₂.Rx.
  - ack₂ is replaced with ack₂.Tx.

• Scheduler automata. Remains the same.

The original four automata are repeated in Figure 3 for convenience.
4 Approach

To derive a relation between the channel delay ($\delta$) and the time difference ($c$) between controller signals, I adapted the original HyTech input files to accommodate for the inclusion of a communication channel between the two sensors and the controller. The modified input files, one for a full-duplex channel and a second for a half-duplex channel, appear in the appendix at the end of this report.

First, I tried to keep $\delta$ as a HyTech-parameter and let HyTech find the conditions on $\delta$ such as $c$ does not exceed some threshold value, but HyTech never terminates with an answer (for both channel models, full- and half-duplex, and both forward and backward reachability analysis). The operating system, which was RedHat Linux, had to kill the HyTech process after more than one hour of execution, most probably, because of unbounded memory buildup. Next, I decided to assign multiple values to $\delta$ and for every value to obtain the condition on $c$ using HyTech. This appears in the first lines, the statements `define(delta, 72/10)` and `define(delta, 90)`, in each of the input files, respectively.

5 Results

I executed HyTech with different values of $\delta$ and for both cases, full- and half-duplex channel models. Figure 4 summarizes these results. I chose to plot $\delta$ versus $c$ and not the other way; therefore, one interested in a specific value of $c$ to meet control timing constraints, may choose a communication channel with appropriate value of $\delta$.

Here, I just make some apparent yet interesting comments on these results without trying to explain them. Explanations need more investigations and perhaps deriving some equations, all of this work is
beyond the scope of this project.

5.1 Comments

- In full-duplex mode, as $\delta$ increases, $c$ increases linearly with slope of 0.5 until $\delta$ reaches 8.5. After that, small changes in $\delta$ cause $c$ to increase very dramatically. When $\delta$ reaches 10, HyTech does not terminate with an answer and for all of subsequent values of $\delta$.

- In half-duplex mode, we can differentiate the following regions of operation:
  1. $0 \leq \delta < 0.3$. $c$ increases linearly as $\delta$ increases with slope of 0.5.
  2. $\delta = 0.3$. There is jump decrease in $c$’s value from 11 to 3.
  3. $0.3 \leq \delta < 9$. $c$ increases linearly as $\delta$ increases with slope of 1. However, for values of $\delta$ between 4.5 and 5.5, HyTech does not terminate with an answer.
  4. $\delta = 9$. There is jump increase in $c$’s value from 12 to 32.
  5. $9 \leq \delta < 10$. $c$ increases linearly as $\delta$ increases with slope of 0.5.
  6. $10 \leq \delta < 14$. $c$ increases linearly as $\delta$ increases with slope of less than 0.5.
  7. $\delta \geq 14$. HyTech does not terminate with an answer.

- The last and the most surprising result is that for the same value of $\delta$, the value of $c$ in full-duplex mode is greater than its counterpart in half-duplex. the difference becomes very large for values of $\delta$ greater than 9.

5.2 Conclusions and Recommendations

From the last comment above, we recommend the use of half-duplex over the full-duplex channel to achieve lower delays in the control signal.
A Full-duplex channel model

define(delta, \( \frac{72}{10} \))

var
\( y_1 \), \( y_2 \), \( x_1 \), \( x_2 \), \( z \), \( t_1 \), \( t_2 \)
: stopwatch;
c
: clock;
alpha
: parameter;

-- ------------------------------------------------------------------ *)

automaton sensor_1
synclabs : request_1, read_1, data_1_Tx, ack_1_Rx;

initially done & \( y_1 = 6 \);

loc done: while \( y_1 \leq 6 \) wait \{ dy_1 = 1 \}
when \( y_1 \geq 6 \) sync request_1 goto read;

loc read: while True wait \{ dy_1 = 0 \}
when True sync read_1 do \{ y_1' = 0 \} goto waiting;

loc waiting: while \( y_1 \leq 4 \) wait \{ dy_1 = 1 \}
when \( y_1 \geq 4 \) sync request_1 goto read;
when asap sync data_1_Tx goto send;

loc send: while True wait \{ dy_1 = 0 \}
when True do \{ y_1' = 0 \} sync ack_1_Rx goto done;
end

-- ------------------------------------------
automaton sensor_2
synclabs : request_2, read_2, data_2_Tx, ack_2_Rx;

initially done & y2=6;

loc done: while y2<=6 wait {dy2 = 1}
   when y2>=6 sync request_2 goto read;

loc read: while True wait {dy2 = 0}
   when True do {y2' =0} sync read_2 goto waiting;

loc waiting: while y2<=8 wait {dy2=1}
   when y2>=8 sync request_2 goto read;
   when asap sync data_2_Tx goto send;

loc send: while True wait {dy2=0}
   when True do {y2' = 0} sync ack_2_Rx goto done;
end

-- -------------------------------------------------

automaton scheduler
synclabs : request_1, read_1, request_2, read_2;

initially idle;

loc idle: while True wait {dx1=0,dx2=0}
   when True sync request_1 do {x1' = 0} goto sensor1;
   when True sync request_2 do {x2' = 0} goto sensor2;

loc sensor1: while x1<=11/10 wait {dx1=1,dx2=0}
   when x1>=1/2 sync read_1 goto idle;
   when True sync request_2 do {x2'=0} goto sensor2_wait1;

loc sensor2: while x2<=2 wait {dx2=1,dx1=0}
   when x2>=3/2 sync read_2 goto idle;
   when True sync request_1 do {x1'=0} goto sensor2_wait1;

loc sensor2_wait1: while x2<=2 wait {dx2=1,dx1=0}
   when x2>=3/2 sync read_2 goto sensor1;
automaton controller
synclabs : data_1_Rx, expire_1, ack_1_Tx,
data_2_Rx, expire_2, ack_2_Tx,
signal;

initially rest & c=0;

loc rest: while True wait {dz=0}
when True sync data_1_Rx do {z'=0} goto rec1_1;
when True sync data_2_Rx do {z'=0} goto rec2_2;

loc rec1_1: while z<=1 wait {dz=1}
when z>=9/10 sync ack_1_Tx goto wait2;

loc wait2: while z<=10 wait {dz=1}
when z>=10 sync expire_2 goto rest;
when True sync data_2_Rx do {z'=0} goto rec1_2;

loc rec1_2: while z<=1 wait {dz=1}
when z>=9/10 sync ack_2_Tx do {z'=0} goto compute;

loc rec2_2: while z<=1 wait {dz=1}
when z>=9/10 sync ack_2_Tx goto wait1;

loc wait1: while z<=10 wait {dz=1}
when z>=10 sync expire_1 goto rest;
when True sync data_1_Rx do {z'=0} goto rec2_1;

loc rec2_1: while z<=1 wait {dz=1}
when z>=9/10 sync ack_1_Tx do {z'=0} goto compute;

loc compute: while z<=56/10 wait {dz=1}
when z>=36/10 sync signal do {c'=0} goto rest;

end
-- -----------------------------------------------------

automaton dataCh
synclabs : data_1_Tx, data_2_Tx, data_1_Rx, data_2_Rx;
initially idle;

loc idle: while True wait {dt1=0}
when True  sync data_1_Tx do {t1’ = 0} goto transmitting_1;
when True  sync data_2_Tx do {t1’ = 0} goto transmitting_2;

loc transmitting_1: while t1<=delta wait {dt1=1}
when t1>=delta  sync data_1_Rx do {t1’ = 0} goto idle;

loc transmitting_2: while t1<=delta wait {dt1=1}
when t1>=delta  sync data_2_Rx do {t1’ = 0} goto idle;
end
-- ----------------------------------------------

automaton ackCh
syndoc : ack_1_Tx, ack_2_Tx, ack_1_Rx, ack_2_Rx;

initially idle;

loc idle: while True wait {dt2=0}
when True  sync ack_1_Tx do {t2’ = 0} goto transmitting_1;
when True  sync ack_2_Tx do {t2’ = 0} goto transmitting_2;

loc transmitting_1: while t2<=delta wait {dt2=1}
when t2>=delta  sync ack_1_Rx do {t2’ = 0} goto idle;

loc transmitting_2: while t2<=delta wait {dt2=1}
when t2>=delta  sync ack_2_Rx do {t2’ = 0} goto idle;
end
-- ----------------------------------------------

var init_reg, bad_reg, reached_reg : region;

bad_reg := c >= alpha;
init_reg := loc[sensor_1] = done & y1=6 & loc[sensor_2] = done & y2=6 &
loc[scheduler] = idle & loc[controller] = rest & c=0 &
loc[dataCh]=idle & loc[ackCh]=idle;
reached_reg := reach forward from init_reg endreach;

prints "Condition to get into the bad state: ";
print omit all locations
hide non_parameters in reached_reg & bad_reg endhide;
B  Half-duplex channel model

define(delta,90)

var
  y1,
  y2,
  x1,
  x2,
  z,
  t1
  : stopwatch;

  c
  : clock;

alpha
  : parameter;

-- -------------------------------------------------------------- *)

automaton sensor_1
synclabs : request_1, read_1, data_1_Tx, ack_1_Rx;

initially done & y1=6;

loc done: while y1<=6 wait {dy1 = 1}
  when y1>=6 sync request_1 goto read;

loc read: while True wait {dy1 = 0}
  when True sync read_1 do {y1'=0} goto waiting;

loc waiting: while y1<=4 wait {dy1=1}
  when y1>=4 sync request_1 goto read;
  when asap sync data_1_Tx goto send;

loc send: while True wait {dy1=0}
  when True do {y1’ = 0} sync ack_1_Rx goto done;
end

-- -----------------------------------------------
automaton sensor_2
synclabs : request_2, read_2, data_2.Tx, ack_2.Rx;

initially done & y2=6;

loc done: while y2<=6 wait {dy2 = 1}
when y2>=6 sync request_2 goto read;

loc read: while True wait {dy2 = 0}
when True do {y2’ =0} sync read_2 goto waiting;

loc waiting: while y2<=8 wait {dy2=1}
when y2>=8 sync request_2 goto read;
when asap sync data_2.Tx goto send;

loc send: while True wait {dy2=0}
when True do {y2’ = 0} sync ack_2.Rx goto done;
end

-- -------------------------------------------------

automaton scheduler
synclabs : request_1, read_1, request_2, read_2;

initially idle;

loc idle: while True wait {dx1=0,dx2=0}
when True sync request_1 do {x1’ = 0} goto sensor1;
when True sync request_2 do {x2’ = 0} goto sensor2;

loc sensor1: while x1<=11/10 wait {dx1=1,dx2=0}
when x1>=1/2 sync read_1 goto idle;
when True sync request_2 do {x2’=0} goto sensor2_wait1;

loc sensor2: while x2<=2 wait {dx2=1,dx1=0}
when x2>=3/2 sync read_2 goto idle;
when True sync request_1 do {x1’=0} goto sensor2_wait1;

loc sensor2_wait1: while x2<=2 wait {dx2=1,dx1=0}
when x2>=3/2 sync read_2 goto sensor1;
automaton controller
synclabs : data_1_Rx, expire_1, ack_1_Tx,
data_2_Rx, expire_2, ack_2_Tx,
signal;

initially rest & c=0;

loc rest: while True wait {dz=0}
when True sync data_1_Rx do {z'=0} goto rec1_1;
when True sync data_2_Rx do {z'=0} goto rec2_2;

loc rec1_1: while z<=1 wait {dz=1}
when z>=9/10 sync ack_1_Tx goto wait2;

loc wait2: while z<=10 wait {dz=1}
when z>=10 sync expire_2 goto rest;
when True sync data_2_Rx do {z'=0} goto rec1_2;

loc rec1_2: while z<=1 wait {dz=1}
when z>=9/10 sync ack_2_Tx do {z'=0} goto compute;

loc rec2_2: while z<=1 wait {dz=1}
when z>=9/10 sync ack_2_Tx goto wait1;

loc wait1: while z<=10 wait {dz=1}
when z>=10 sync expire_1 goto rest;
when True sync data_1_Rx do {z'=0} goto rec2_1;

loc rec2_1: while z<=1 wait {dz=1}
when z>=9/10 sync ack_1_Tx do {z'=0} goto compute;

loc compute: while z<=56/10 wait {dz=1}
when z>=36/10 sync signal do {c'=0} goto rest;

end
ack_1_Tx, ack_2_Tx, ack_1_Rx, ack_2_Rx;

initially idle;

loc idle: while True wait {dt1=0}
when True sync data_1_Tx do {t1' = 0} goto transmitting_1;
when True sync data_2_Tx do {t1' = 0} goto transmitting_2;
when True sync ack_1_Tx do {t1' = 0} goto transmitting_3;
when True sync ack_2_Tx do {t1' = 0} goto transmitting_4;

loc transmitting_1: while t1<=delta wait {dt1=1}
when t1>=delta sync data_1_Rx do {t1' = 0} goto idle;

loc transmitting_2: while t1<=delta wait {dt1=1}
when t1>=delta sync data_2_Rx do {t1' = 0} goto idle;

loc transmitting_3: while t1<=delta wait {dt1=1}
when t1>=delta sync ack_1_Rx do {t1' = 0} goto idle;

loc transmitting_4: while t1<=delta wait {dt1=1}
when t1>=delta sync ack_2_Rx do {t1' = 0} goto idle;
end

-- -----------------------------------------------------

var init_reg, bad_reg, reached_reg : region;

bad_reg := c >= alpha;
init_reg := loc[sensor_1] = done & y1=6 & loc[sensor_2] = done & y2=6 &
loc[scheduler] = idle & loc[controller] = rest & c=0 &
loc[Channel]=idle;
reached_reg := reach forward from init_reg endreach;

prints "Condition to get into the bad state: ";
print omit all locations
hide non_parameters in reached_reg & bad_reg endhide;