Control over CAN and Flexray

Embedded Control Systems

Sai Krishna Kalluri & Snorri Stefansson

April 2, 2017

Contents

Chapter 1

Part 1

1.1 Introduction

1.2 Response Time analysis

1.2.1 Fixed-Priority Non-Preemptive

CAN messages follow fixed-priority non-preemptive protocol. The response time analysis for fixed-priority non-preemptive messages can be computed as follows.

BlockingTime

Blocking time of a message is defined as the time period for which the message can be blocked by a lower priority message. Therefore blocking time can be computed by finding the maximum of execution of times of all the messages having lower priority than the current message. It can be computed as shown in equation 1.1.

$$B_{mi} = \max_{k \epsilon lp(m_i)} (c_i) \tag{1.1}$$

BusyPeriod

Busy Period is defined as the maximum time at which the execution of the message gets completed. It includes the blocking time, delay due to execution of higher priority messages and the message execution time. Busy period of message m_i with priority i can be computed from the equation 1.4 with initialization condition shown in equation 1.2 and termination condition as shown in equation 1.3.

$$t_{mi}^{k+1} = B_{mi} + \sum_{\forall m \epsilon hp(m_i) \cup m_i} \lceil \frac{t_{mi}^{k}}{p_m} \rceil c_m i \tag{1.2}$$

$$t_{mi}^{0} = c_{mi} \tag{1.3}$$

$$t_{mi}^{k+1} = t_{mi}^{k} \tag{1.4}$$

ResponseTime

If the busy period of the message is less than the period of the message, then the response time of the message is equal to busy period. But if busy period is greater than the period of the message, multiple messages arrive in this instance and response time is computed as the maximum of response times of each individual instances. The number of instances can be computed from equation 1.5. The waiting time and response time of instance q can be computed using equations 1.6 and equation 1.7 respectively. The initialization

and termination conditions for each instance are governed by equations 1.8and 1.9 respectively. The total response time can be computed using equation 1.10.

$$Q_{mi} = \lceil \frac{t_{mi}}{p_{mi}} \rceil \tag{1.5}$$

$$w_{mi}^{k+1}(q) = B_{mi} + qc_{mi} + \sum_{\forall m \epsilon hp(m_i)} \lceil \frac{w_{mi}^{k}(q) + \tau_b it}{p_m} c_m \rceil \tag{1.6}$$

$$R_{mi}(q) = w_{mi}(q) - qp_{mi} + c_{mi} \tag{1.7}$$

IniTialization and Termination:

$$w_{mi}^{0}(q) = B_{mi} + qc_{mi} \tag{1.8}$$

$$w_{mi}^{k+1}(q) = w_{mi}^{k}(q) \tag{1.9}$$

Total Response time of the message:

$$R_{mi} = \max_{q=0 to Q_{mi}} (R_{mi}(q)) \tag{1.10}$$

1.2.2 Fixed-Priority Preemptive

Unlike the above case messages or tasks can be preempted in Fixed-Priority preemptive case. The processors PU1 and PU2 employ FPP for their task management. The response time of FPP can be calculated as below.

ResponseTime

The waiting time and response time of each instance can be calculated from equations 1.11 and 1.12. The initialization and termination conditions for each task can be obtained from equations 1.13 and 1.14. The response time is nothing but the settling value of the response time.

$$w_{mi}^{k+1}(q) = w_{mi}^{k}(q) + \sum_{\forall m \epsilon hp(m_i)} \lceil \frac{R_{mi}^{k}(q)}{p_m} c_m \rceil \tag{1.11}$$

$$R_{mi}(q) = w_{mi}(q) + c_{mi} \tag{1.12}$$

IniTialization and Termination:

$$w_{mi}^{0}(q) = 0; \tag{1.13}$$

$$R_{mi}^{k+1}(q) = R_{mi}^{k}(q) \tag{1.14}$$

Table 1.1: By running the Matlab script ResponsetimeAnylsis_FPP.m with the different parameters given for PU1 and PU2 these response times are obtained for each of the tasks. These files are then delivered as PU1.m PU2.m

PU1	T_1	T_2	T_3	T_4 (T_s)
Matlab (ms)	0.1	2.1	4.1	7.2

PU2	T_5	T_6	T_7	T_8
Matlab (ms)	6	3	9	5

Table 1.2: Response times for the CAN bus messages

CAN	m_2	m_1	m_3	m_8
Matlab (ms)	2	3	4	4

1.3 System Model Derivation and Control Parameter Design

The objectives for the controller is to properly control the given system with a set of design parameters. These parameters, shown in Table

There are namely a number of steps taken until all parameters can be considered to satisfy the performance constraints of the controller. These basic steps can be seen in the following list.

1. Step 1: Derive the system model. Determine the A, B and C matrices in relation to all constants by hand calculations. We are already provided with the matrices in assignment description, hence no need to derive.Insert this into Matlab.
2. Step 2: Derive the values for sampling period and sensor-to-actuator delay based on our system design.
3. Step 3: Choosing desired poles according to the given requirements. Verfiying the controllability of the system for the choosen values. Computing the controller Feed-Forward and Feed-Back gains F and K. Inserting these calculations into MATLAB. More about this step can be found in Section
4. Step4: Design K and F values, compute current input activation(u) and outputs(x) from equations, insert these calculations to MATLAB and simulate the system.
5. Step5: Apply multi-objective genetic algorithm on the above system, with pole positions as parameters, and settling-time and maximum input voltage as our objectives. Obtain pareto optimal points satisfying design constraints.

Table 1.3: Constants and design parameters referenced to in calculations

Symbol	Description	Value	Unit
θ	Rotor position	-	rad
i_m	Motor current	-	A
J	Inertia	$3.2284 \cdot 10^{-6}$	Kgm^2
b	Friction coefficient	$3.5077 \cdot 10^{-6}$	Nms/rad
R	Armature Resistance	4	Ohm
L	Inductance of Motor Winding	$2.75 \cdot 10^{-6}$	H
K_m	Motor constant	0.0274	Nm/A
K	Feed-BackGain	-	-
F	Feed-Forward Gain	-	-

1.3.1 Controller Structure

The system can be described with the second order differential equation shown in Equation 1.15 as given in the assignment description. Equation 1.16 and Equation 1.17 describe the statespace representation of our system. From these equations the Feedback- and Feedforward gain can be determined in relations to the constants in the system.

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \theta \\ \dot{\theta} \\ i \end{bmatrix} = \begin{bmatrix} \theta \\ \omega \\ i \end{bmatrix} \quad \text{and} \quad \dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \\ \dot{i} \end{bmatrix} \tag{1.15}$$

$$\dot{\boldsymbol{x}} = \mathbf{A}\boldsymbol{x} + \mathbf{B}u \text{ and } y = \mathbf{C}\boldsymbol{x} \qquad \text{where} \quad \text{input: } u = V \quad \text{, output: } y = i \tag{1.16}$$

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & \frac{-b}{J} & \frac{K_m}{J} \\ 0 & \frac{-K_m}{L} & \frac{-R}{L} \end{bmatrix} \begin{bmatrix} \theta \\ \omega \\ i \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{L} \end{bmatrix} V \quad \text{and} \quad y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \omega \\ i \end{bmatrix} \tag{1.17}$$

1.3.2 Controller Design

In this subsection we discuss the method for designing the controller design parameters FeedbackGain-K and FeedForwardGain-F. In this problem we have the scenario where $D_c < h$. Since our controller operates in discrete sample time, we start with converting our continuous system as described in equation 1.18 into discrete domain. Therefore on applying ZOH sampling with period h_c and constant sensor-actuator Delay D_c we achieve the equation 1.19 for discrete sample time system. From 1.19 we can notice that the next output not only depends on current input but also on previous input. Hence, we simplify the system and get it into standard form representing equation 1.20 by invoking equation1.21. After applying above simplification input $u[k]$ can be represented in terms of controller gains using equation1.22. and matrices Φ ,C are converted to corresponding augmented matrices Φ_{aug},C_{aug}, whereas Γ_1,Γ_2 are converted to single augmented matrix Γ_{aug}.

$$\dot{x} = Ax + Bu \quad \text{and} \quad y = Cx \tag{1.18}$$

$$x[k+1] = \phi x[k] + \Gamma_1(D_c)u[k-1] + \Gamma_0(D_c)u[k] \quad \text{and} \quad y[k] = Cx[k] \tag{1.19}$$

$$x[k+1] = \phi z[k] + \Gamma_{aug}u[k] \quad \text{and} \quad y[k] + C_{aug}z[k] \tag{1.20}$$

$$z[k] = \begin{bmatrix} x[k] \\ u[k-1] \end{bmatrix} \tag{1.21}$$

$$u[k] = Kz[k] + Fr \tag{1.22}$$

$$K = -[\ 0\ 0\ \cdots\ 1\ ]\gamma_{aug}^{-1}H(\phi_{aug}) \tag{1.23}$$

$$F = \frac{1}{C_{aug}(I - \phi_{aug} - \Gamma_{aug}K)^{-1}\Gamma_{aug}} \tag{1.24}$$

Before deriving the controller gains it is important to verify if the system is controllable under given configuration. This can be verified by calculating $det(\gamma_{aug})$. If the determinent is not equal to 0, then system is controllable. After verifying the controllability of the system K can be derived by applying Ackermanns formula described in equation 1.23 and F can be derived from equation 1.24 . However, one can notice the matrix $H(\Phi_{aug})$ depends on the desired poles which in turn depends on the design requirements. The desired poles alter the frequency spectrum of the transfer function of the system and play a significant role in controlling the behavariour of output parameters of the system. Therefore various design requirements such as Overshooot, settling time, boundary ranges of the parameters in the system depends on the desired poles and thus in turn influence controller gains K and F. Following design, work flow has been developed in MATLAB to derive pareto optimal points(pole locations) satisfying design constraints using multi-objective genetic algorithm and one of the optimal point is selected for simluation.

1.4 Simulation and Design Decisions

Foll0wing are the design decisions taken before simulating the system.

1. Employing Rate-Monotonic -Scheduling (RMS). That is the tasks with lower period gets higher priority. This ensures Task T_s and message m_s have highest priorities and thus reducing sensor-actuator delay. Therefore Sensor-to-actuator delay can be computed as

$$D_c = T_s + m_s + 2S + T_c a$$

   This results in sensor-actuator delay of 8.1ms. The TDMA cycle of PU3 is 10ms($> D_c$). Therefore we can have a sampling period of 10ms.

2. Simulate the above system using above design parmeters h_c and D_c.Apply multi-objective genetic algorithm on the above system, with pole positions as parameters, and settling-time and maximum input voltage as our objectives. The design constraints are settling time less than 400ms and input voltage less than 1v. Obtain pareto optimal points satisfying design constraints. The lower and upper bounds used for pole positions are 0 and 1 respectively.

1.5 Results

The input data (period, deadline ,wcet and priority) for both the processors and CAN bus is shown in the left part of the figure 2.7. The priorities for tasks are chosen according to Rate-Monotonic-Scheduling (RMS). The simulation for response times for PU1 and PU2 is computed using $'ResponseTimeAnanlysis\_FPP.m'$ which performs the calculations shown described in section 1.2.2 while the response time for CAN bus is computed using $'ResponseTimeAnanlysis\_FPNP.m'$ which performs the calculations shown described in section 1.2.2. The response times for tasks in PU1 and PU2 are shown in Table 1.1 and the response time of CAN messages is shown in Table 1.2. The pareto front and the score histogram for optimal pole positions is shown in figure 1.1. The pareto optimal points are presented in Table 1.5. The middle point is taken for simulation and the plots for rotor position and input voltage are shown in figure 1.2.

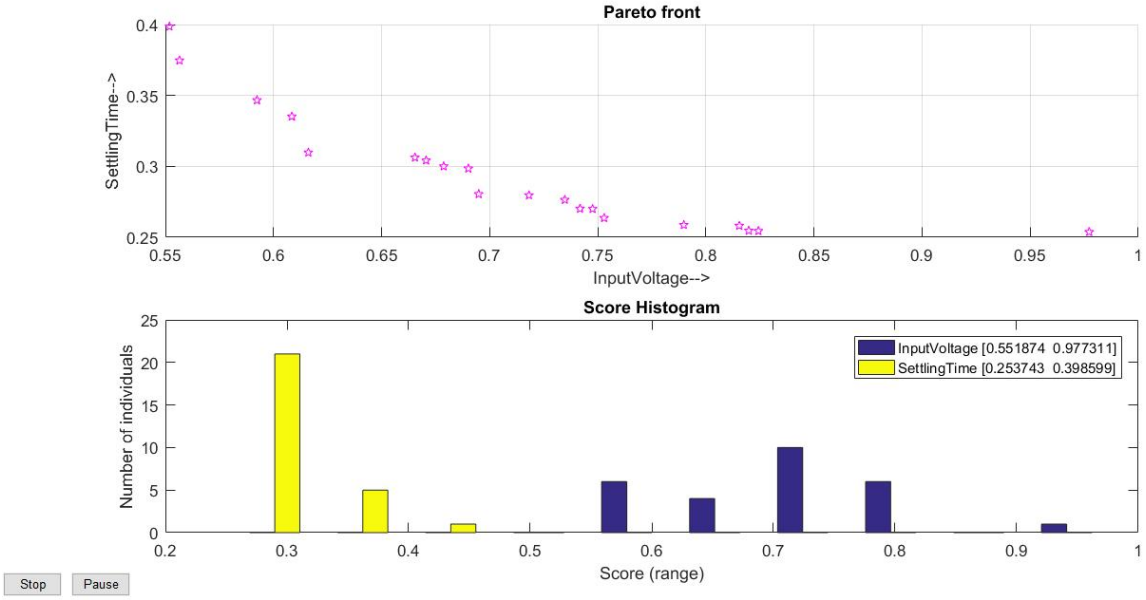

Figure 1.1: Pareto Plot and score histogram for various pole positions with settling time and input voltage as objectives

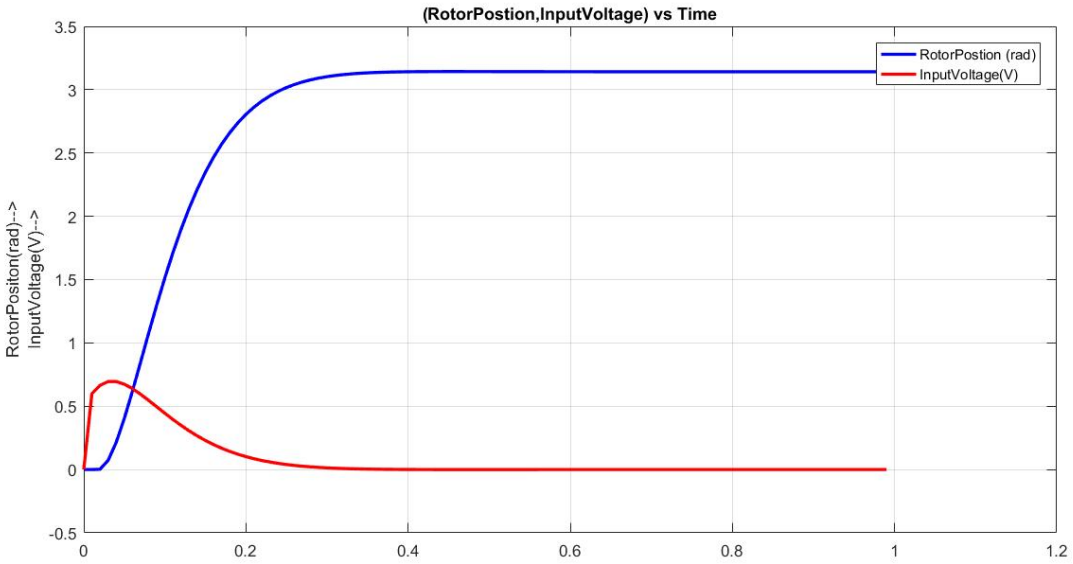

Figure 1.2: Change in Rotor Position and Input Voltage with time (for the 10th pareto point)

InputVoltage(V)	SettlingTime(sec)	P1	P2	P3	P4
0.55187	0.3986	0.042356	0.79712	0.86895	0.032116
0.55651	0.37465	0.033305	0.79853	0.87171	0.039792
0.59234	0.34662	0.0058031	0.82007	0.84669	0.036008
0.60848	0.33514	0.036173	0.79947	0.85949	0.032497
0.6161	0.30976	0.019745	0.82096	0.84783	0.029296
0.66542	0.30623	0.043944	0.79782	0.83793	0.01936
0.67059	0.30415	0.02643	0.80271	0.83055	0.023413
0.6787	0.3	0.019514	0.80909	0.82646	0.01884
0.6901	0.2985	0.026171	0.80598	0.82174	0.018069
0.69489	0.28047	0.023082	0.80557	0.82825	0.02835
0.71815	0.27962	0.017678	0.80943	0.81624	0.0096658
0.73472	0.2764	0.026266	0.79633	0.82358	0.017902
0.74179	0.27011	0.030858	0.80199	0.81578	0.015076
0.7476	0.26996	0.019604	0.79668	0.81833	0.0062062
0.75291	0.26361	0.023345	0.79621	0.81719	0.028087
0.78977	0.25864	0.027588	0.79019	0.80703	0.020728
0.81545	0.25809	0.020387	0.7722	0.81093	0.033555
0.81984	0.25468	0.0095985	0.79821	0.7961	0.013229
0.8243	0.25444	0.010459	0.7929	0.79613	0.013602
0.97731	0.25374	0.012836	0.78696	0.75179	0.024935

Table 1.4: Pareto Points (Pole locations) for Input Voltage and Settling Time

Chapter 2

Part 2

2.1 Introduction

Analyzing and confirming broad analysis done on paper and in Matlab allows us to confirm the hypothesis made about the behavior of the CAN bus and its specific tasks from Part 1. This analyses is done in Inchron, which allows us to explore the real-time behavior of this embedded system in full detail. Some exploration on how the system should work has already been done in Part 1, Add Table reference to periods and priorities , this is used here by feeding the settings of the embedded system to the tool. The settings included is the hierarchy of the Processing Units (PUs) and their tasks which each have different periods, execution time and priority. The tree view, Figure 2.1, shows the details of the hierarchy from the Inchron's perspective.

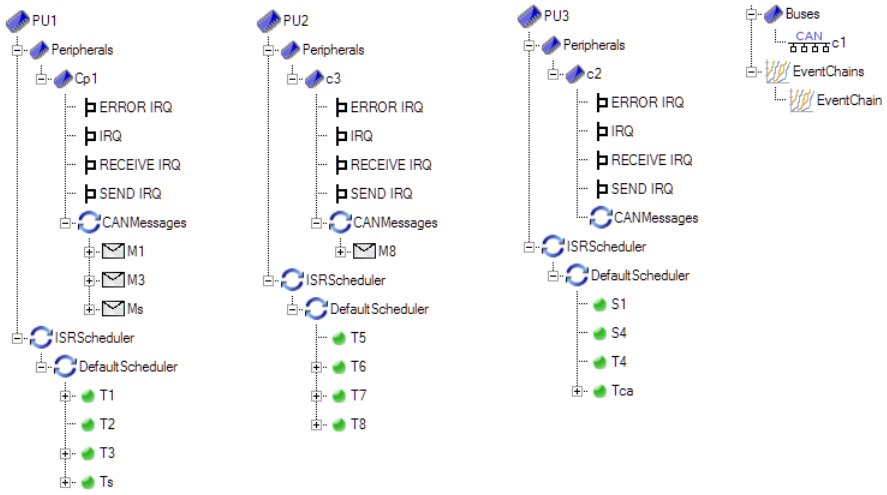

Figure 2.1: This shows the coarse grain hierarchy of the system ported into Inchron for verification and simulation of the real time components of our embedded system.

2.2 Response Time analysis

2.2.1 Response time analysis per processing unit

For this analysis all details have to be imported to Inchron as stated earlier. Now paying special attention to the messages which transfer the packets between controllers which make a full system.

When validating and simulating the model with the settings mentioned in Part 1 and Figure 2.1 and 2.5 we can observer the response times for each task when inserting the Worst Case Execution Time into the tool. Next each processing unit and their Worst Case Response Time (WCRT) will be investigated and then compared to the Matlab model (Response Time analysis). The resulting figures were obtained after several tries with various settings until the correct configuration of the tool was found, e.g. setting the schedule as preemptive was not set as it was not found in the first try.

PU1

For PU1 a fixed preemptive priority scheme with four tasks T1,T2,T3 and Ts. Now when validating the model, Figure 2.2 is obtained showing the results and they are compared to the Matlab response times in Table 2.1.

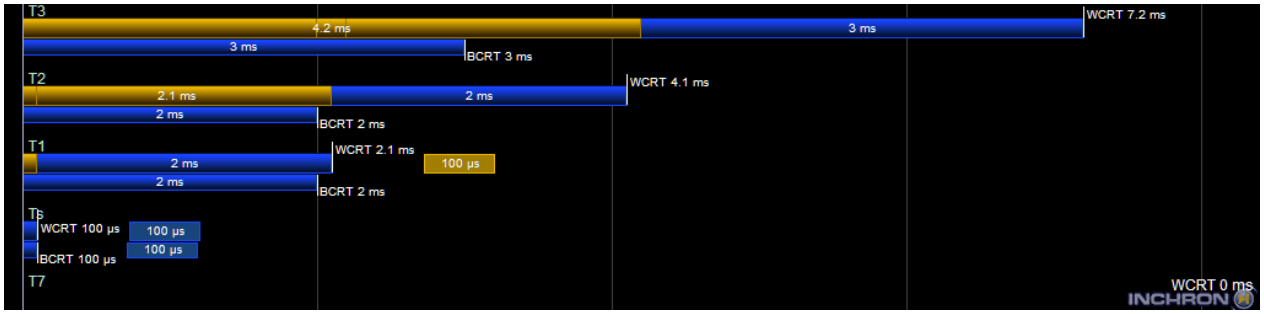

Figure 2.2: Showing the WCRT analysis for PU1. The Results can be read from each horizontal bar.

PU2

Now PU2, has the same sheme as PU1, fixed preemptive priority, with four tasks T5,T6,T7 and T8. Now when validating the model, Figure 2.3 is obtained showing the results and they are compared to the Matlab response times in Table 2.1.

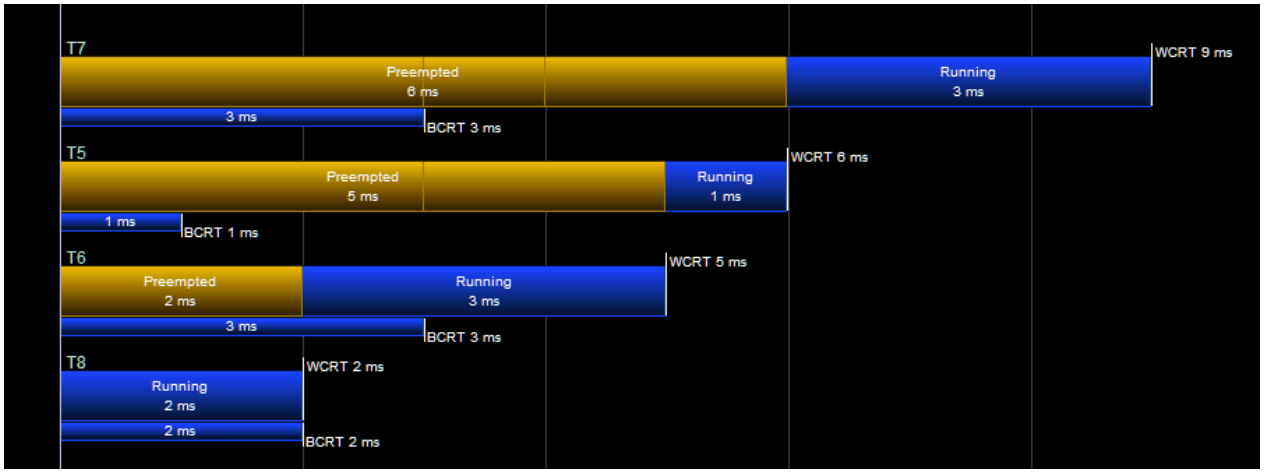

Figure 2.3: Showing the WCRT analysis for PU2. The Results can be read from each horizontal bar.

PU3

Allthough PU3 has a time division multiplexing (TDM) its tasks will still be analyzed here, shown in Figure 2.4. It is important to state that T4 is not a real-time task as it is sending a message *out to the blue* but T_{ca} is an important task, actuating on the sensor value and completing the sensor to actuator delay.

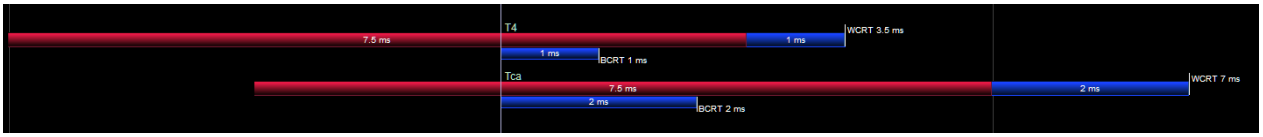

Figure 2.4: Showing the WCRT analysis for PU3 (TDM). The Results can be read from each horizontal bar

Table 2.1: By running the Matlab script ResponsetimeAnylsis_FPP.m with the different parameters given for PU1 and PU2 these response times are obtained. These files are then delivered as PU1.m and PU2.m

PU1	T_1	T_2	T_3	T_4 (T_s)
Matlab (ms)	0.1	2.1	4.1	7.2
Inchron (ms)	0.1	2.1	4.1	7.2

PU2	T_5	T_6	T_7	T_8
Matlab (ms)	6	3	9	5
Inchron (ms)	6	3	9	5

2.2.2 Response time analysis for the CAN bus messages

PU1 and PU2 are the only units within the system that are sending messages, shown for clarity in Figure 2.5, but PU3 contains the computing and actuating task which will receive the m_s message and mark the end of the sensor to actuator delay.

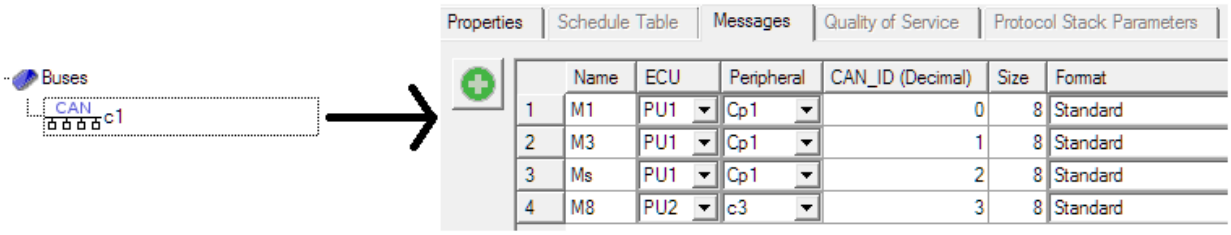

Figure 2.5: All can messages in the system, indicating PU source, individual CAN id and message size in bytes

2.3 Optimization for sensor-to-actuator delay

Now optimizing the sensor-to-actuator delay requires running the CrhonOpt tool within Inchron and setting up objectives for the real time requirements. That will be set to target: event-chain, which is set to be smaller or equal to 5ms.

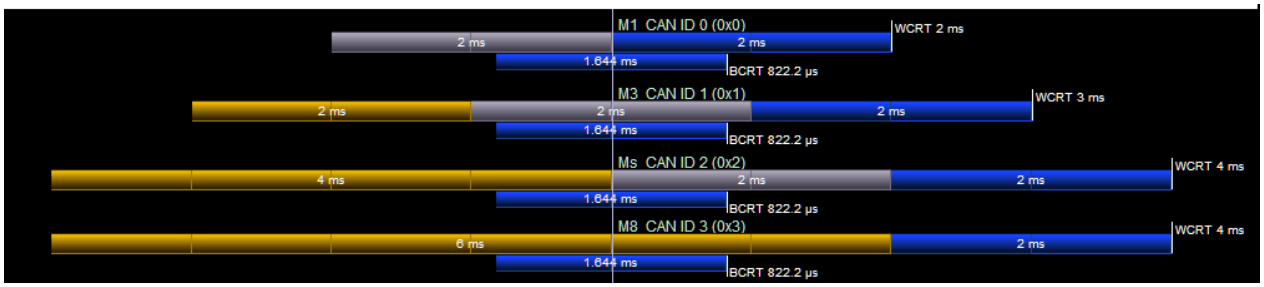

Figure 2.6: Showing the WCRT analysis for the CAN messages. The Results can be read from each horizontal bar and is compared in Table 2.2.

Table 2.2: CAN messages of the system compared from the Inchron tool suite and Matlab. Showing identical results

CAN	m_2	m_1	m_3	m_8
Matlab ms	2	3	4	4
Inchron	2	3	4	4

The initial priorities were selected according to period and execution time, shown in Figure 2.7 on the left, but when running the simulation to enhance the sensor to actuator delay, the tool changed the priorities in PU2, shown in Figure 2.7 on the right. This showed to be an improvement although this did not change the sensor to actuator delay.

```
%% period, deadline, wcet and prioriy

%PU1                    Task    Inchron Priorities
 T = [5, 10, .1, 1;     % Ts    4
      10, 1, 2, 2;      % T1    3
      15, 1, 2, 3;      % T2    2
      20, 5, 3, 4;      % T3    1
     ];

%PU2                    Task  Inchron Priorities
 T = [15, 10, 1, 3;     % T5  2
      10, 1, 3, 2;      % T6  3
      20, 1, 3, 4;      % T7  1
      10, 5, 2, 1;      % T8  4
    ];

% CAN messages                  Inchron Priorities
 T = [5, 2, 1, 1;       % m_s   4
      10, 15, 1, 2;     % m_1   3
      20, 30, 1, 4;     % m_3   2
      10, 10, 1, 3;     % m_8   1
      ];
```

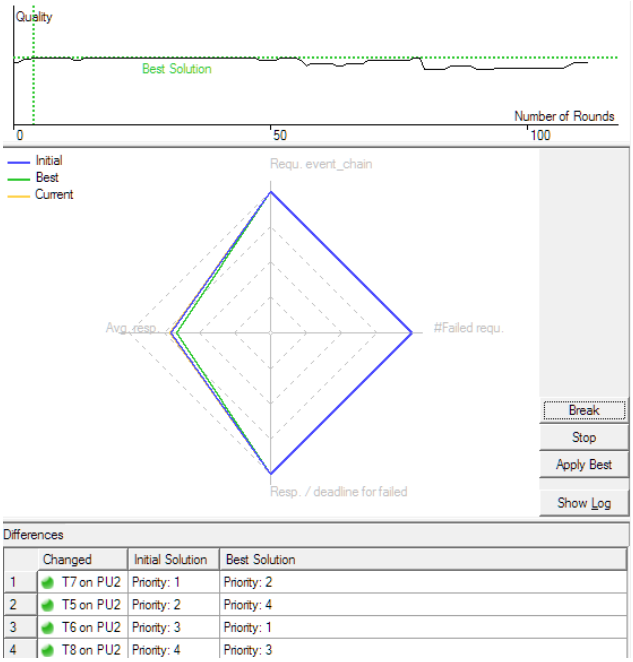

Figure 2.7: On the left, initial priorities used in the Matlab script and on the right the ChronOPT optimization of these parameters.

2.4 Design decision

Design decision are used from Part 1, seen in Section 1.4. To satisfy the design decision made in Part 1 saying that the sensor to actuator delay should be 8.1 ms as mentioned in Section 1.4 as well, some changes have to be made to the Inchron project. Without any changes to the project the Sensor to actuator delay will be 12 ms seen in Figure 2.8, as well notice the blue are which is 4.9 ms which should be 1 ms. So what we did was add a offset of 4.9 ms - 1 ms = 3.9 ms to the T_1 task. This allows the model to act correctly and show the correct sensor to actuator delay of 8.1 ms and also lowering the sampling period to 10 ms.

2.5 System model and Results

Firstly: The Response time analysis for each of the Processing units was performed in Inchron and the timings match the timings obtained from the Matlab model as seen in Table 2.1 as for the CAN messages, seen in Table 2.2. Secondly: Plots from chronVIEW (before and after optimization)
Last: Control system input and output

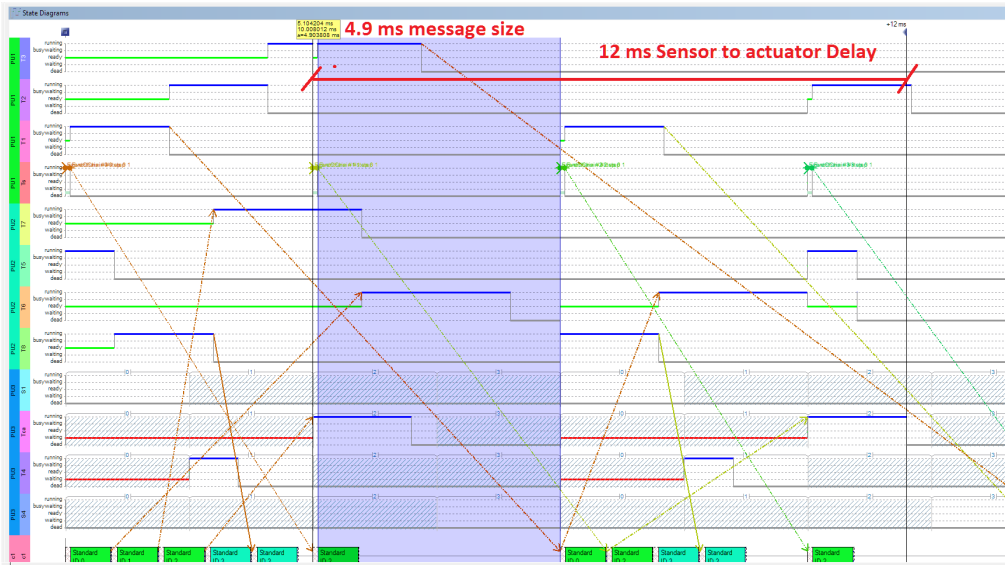

Figure 2.8: The first 20ms (equal to largest period) of the **optimized state diagram** for the CAN model. Showing the sensor to actuator delay and the wrong message time of m_1

2.6 Conclusions

To conclude, the theoretical/Matlab analysis goes hand in hand with the Inchron modeling when used correctly. The tool is very powerful and can count for all design decision made in Part 1. The tool allows for a broad overview of the complicated system as well as troubleshooting more difficult sections of the analysis to maximize the performance of the CAN bus according to the theoretical analysis.

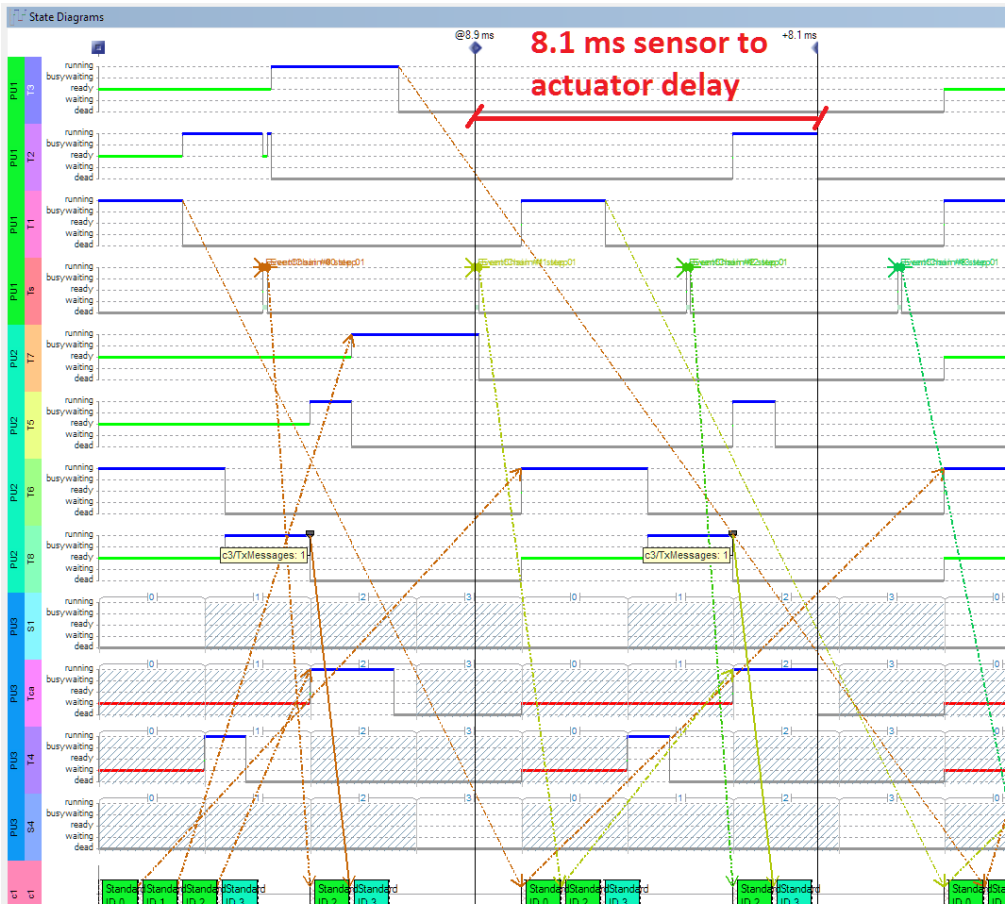

Figure 2.9: The first 20ms (equal to largest period) of the **non optimized state diagram** for the CAN model.

Chapter 3

Part 3

3.1 Introduction

Now describe the setup for the FlexRay modeling part, Figure 3.1 shows the most important given parts and also the tree view in Inchron along with the FlexRay bus connections.

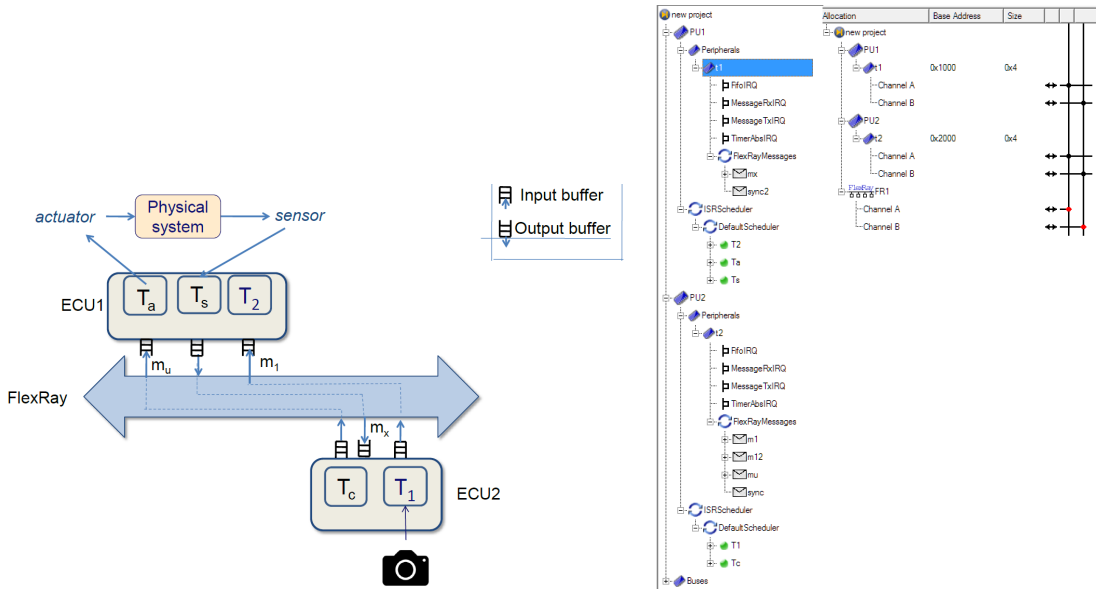

Figure 3.1: On the left, system diagram and on the right the tree view and architecture of the Inchron project.

The main purpose in this part is to firstly create a theoretical analysis to be able to model the FlexRay in Inchron. Then seeing how the implementation is doing in the model and optimize any timings that have room for change.

3.2 Answering the questions

1. *What is the maximum cycle length possible?*

   The given cycle length is 4 ms so that is the maximum cycle length in our case.

2. *What are the maximum possible durations for:*

(a) *static slot?* If considering only static slots with size of 0.5 (largest execution time considered of all messages AND tasks for easy scheduling) then maximum is $\frac{4\ ms}{0.5\ ms} = 8$

(b) *dynamic slot?* Only dynamic considering a constant size of 0.25 ms: $\frac{4\ ms}{0.25\ ms} = 16$

(c) *Network idle time?* This will maintain synchronization between the two ECUs and for each it takes one slot so in total this is taking up two slots $= 2\ slots \cdot 0.5\ ms = 1\ ms$.

3. *Derive the implicitly defined parameters (e.g., no. of minislots).*

4. *Why do you need idle phase within a dynamic slot?* There is some idle time needed to react correctly on the coming messages to dynamic slots.

5. *Does the above parameters conform to FlexRay specification?*

3.2.1 Theoretical analysis versus actual implementation

The theoretical analysis was done on paper with most parameters known and then model in Inchron to validate the behavior. There are obvious differences concerning cold start and such that will differ in actual implementation and in theory. But is is possible to counteract this be at least assuming a difference, not knowing how large the difference will be.

3.3 Design decision

Firstly when starting design of theoretical analysis, it was done on paper to figure out the order of the messages and tasks within the static slots and which should go in the dynamic slot. The largest message would be best put in the dynamic slot, m_1. Then when realizing that the dynamic slots are not behaving correctly in Inchron, then it was decided to use only static slots. Then a problem arose, the size of each static slot is only 200 bytes but m_1 is 290 bytes. Thus splitting it up to two will do the trick.

It is very important to note that communication is included in the theoretical analysis but in Inchron they were skipped to lower the complexity but still there is room for everything and they are encountered for. In Figure 3.2 the analysis on paper is showed for both ECUs and the FlexRay and how the messages are executing.

After applying this setup, exact parameters are shown in the results, to the model it showed that the tasks were staring before their cycle was actually happening. By measuring the exact offset it was possible to add it to all the tasks and messages to make the systems behave correctly.

3.4 Results

Firstly: Solution to the design problem. (Include the parameters you have chosen)
Secondly: from chronVIEW for your design

3.5 Conclusions

The final version of the parameters work as expected counteracting the initial offset. The reason for the initial offset is partially unknown but it is likely caused by the coldstart and obviously the synchronization is not behaving exactly as set to.

3.6 Results

3.7 Conclusion

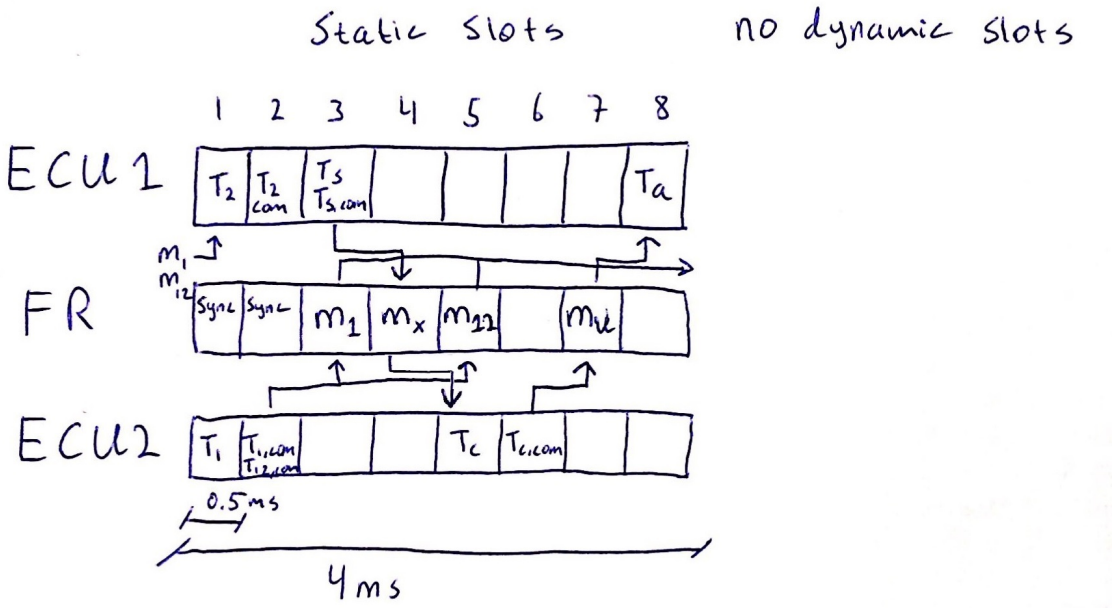
Static slots
no dynamic slots
1 2 3 4 5 6 7 8
ECU 1
T_2
$T_{2\,com}$
T_s
$T_{s\,com}$
T_a
m_1
m_{12}
FR
Sync
Sync
m_1
m_x
m_{22}
m_{ll}
ECU2
T_1
$T_{1,com}$
$T_{1,2,com}$
T_c
$T_{c,com}$
0.5ms
4ms

Figure 3.2:

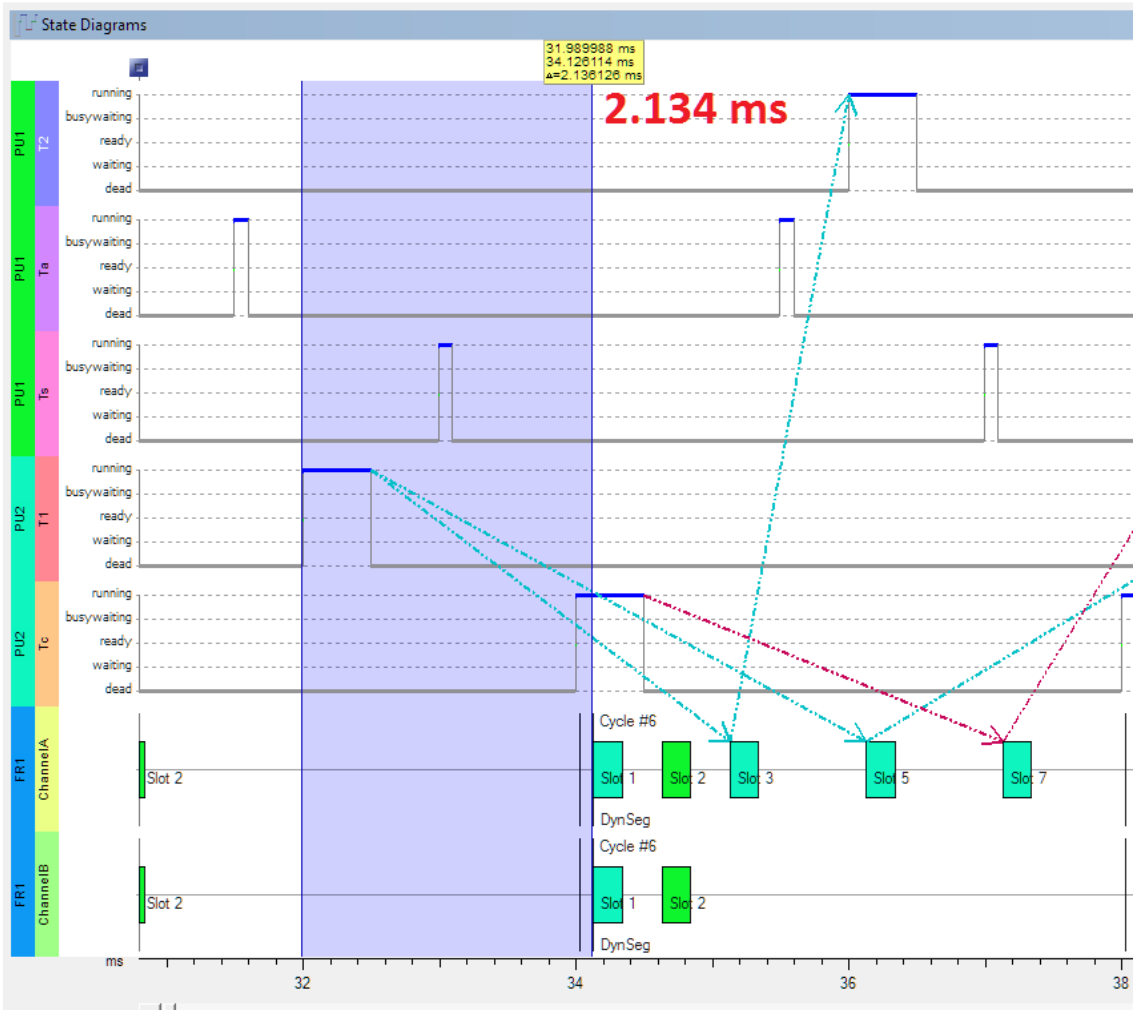

Figure 3.3: Before optimization/offset the tasks are not hitting the start of the cycle, in fact all of the tasks are too early by 2.134 ms as seen in this figure.

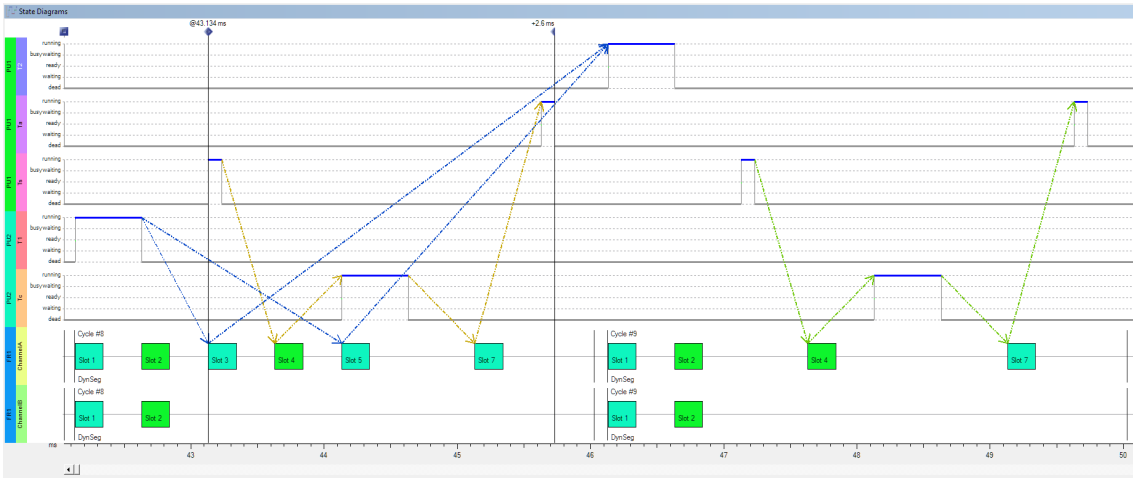

Figure 3.4: