Capita Selecta: Software engineering for automotive systems

Objective

- Architecture Description Languages:
    - Students understand the purpose of an ADL.
    - Students can use a specific ADL for automotive systems.
- Students understand the AUTOSAR method of automotive software engineering.
- Students understand the need for analytical methods for real-time applications.

Context

- AUTOSAR (see presentation)
- Real-time embedded systems (see presentation)
- Automotive market (see presentation)
- ABS – ESP [1]

Assignment

Purpose of ADLs

Skim, mostly the introductions, of [2,3] about Architecture Description Languages. Reflect on this using the following questions:

- What is the purpose of an ADL?
- Why do we need this in automotive software development? Explain your answer with the new trends in the automotive sector.
- What features should an automotive ADL contain? (Reflect on the type of applications, how automotive products are build -> OEM/Tier1/Tier2)

Feature and configuration models

Prerequisites:

Install the axBench[1] plugin in your eclipse environment. We will use this "small" ADL for the modeling part of the assignment. We used a clean install of the "eclipse modeling tool" version Helios SR1.

Background:

[1] http://axbench.isst.fraunhofer.de/

We will make a feature and configuration model of a part of the brake system of a car. Our braking system consists of a conceptual ABS and ESP. For those not familiar with these technologies, look at the documents section for some background information. While ABS is mandatory in all modern cars, the ESP is still an option that the consumer has to choose.

Assignment:

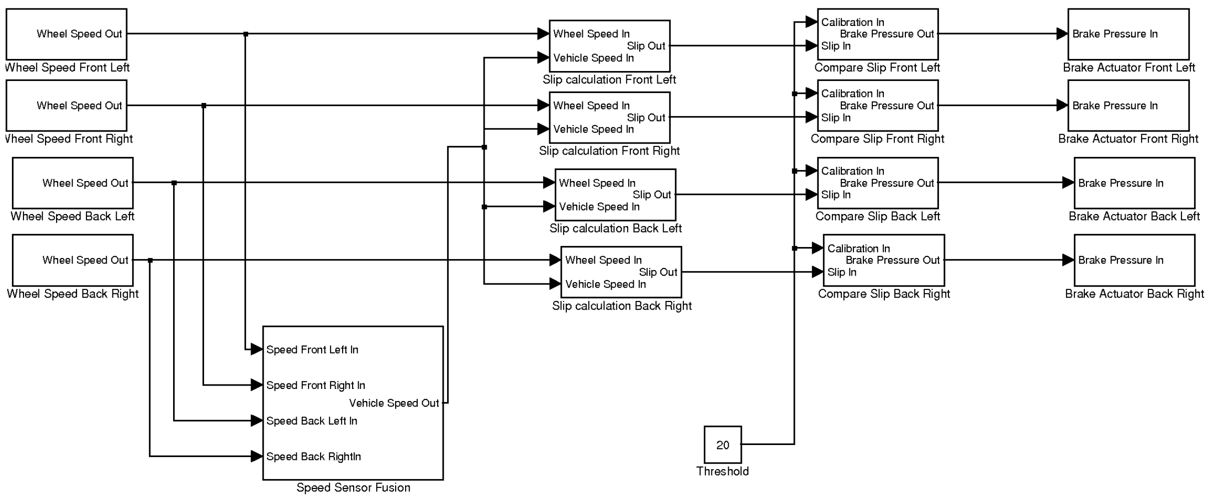
- Implement this in a feature model in the axBench ADL.
- Make 2 configuration models: One with ESP and one without ESP.

Application model

Background:

We briefly discuss the ABS and ESP models. Both models are made in simulink (mathworks). Every block will generate exactly 1 function (runnables). Runnables are called services in the axBench ADL. Since these are runnables, you can choose how to partition them into AUTOSAR "atomic software components". For a quick overview of AUTOSAR, you can always refer to [4].

1. ABS-system:

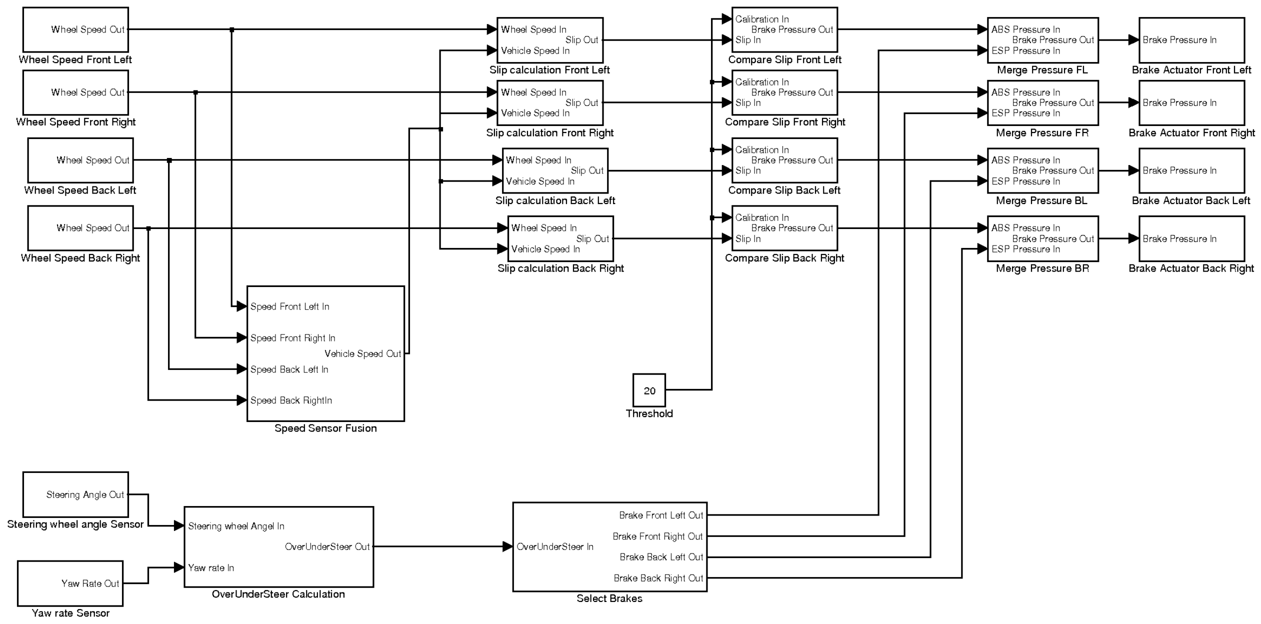


Explanation of the model:

- Wheel Speed (4): These are the sensor components. The sensor outputs a block wave that is linear to the speed of the wheel. This block uses the HW timer to read the pulses and convert them to the engineering values in m/s. The output is a value of type sint32 (signed integer of 32 bits).
- Speed Sensor fusion: The real vehicle speed is obtained by filtering and fusion of the 4 wheel speeds. The output is a value of type sint32.

- Slip calculation: The slip is derived by comparing the real-vehicle speed and the individual wheel speed. The output is a value of type sint16. (signed integer of 16 bits)
- Compare slip: The slip value is compared to a specific threshold. If this threshold is exceeded, the brake has to apply less brake force so the wheel regains grip. The output is a value of type sint16.
- Brake Actuator: The brake actuator regulates the braking valve, increasing or decreasing the braking force.

2. ESP and ABS model:



The model of the ABS is the same as the previous model, only the values of the ESP and ABS are combined to have a single value for applying pressure on the breaks.

- Steering wheel angle sensor: Reads the angle the user wants to steer. This is done using an analog digital convertor. The output is a value of type sint32
- Yaw rate sensor: Reads yaw rate of the vehicle. This is the angular velocity around the axis. This is done using an analog digital convertor. The output is a value of type sint32.
- OverUnderSteer calculation: By subtracting the yaw rate from the steering wheel value we know the amount of under or over steer. The output is a value of type sint16.
- Select brake: Depending on the over or under steer amount, the select function applies the correct amount of braking force on the correct wheel. The output is a value of type sint16.

<u>Assignment:</u>

- Model the components in the axLang ADL. Always use S/R interfaces. Define a single top component without any output ports (the braking system) and instantiate the defined components as submodels and connect them.
- Define a clock component. It has a single output port with a sender/receiver interface. The data-element is a Boolean. Instantiate this component in your braking system. Add a In-port with this interface to your sensor components and connect the clock to the sensor components.
- Define services (runnables).
- Add the traceability links using the feature to application mapping.
- Validate the model and generate the variants. Validate and export the variants to *.dot file for your report. These files can be made into an image file using the graphviz[2] utility. The .axlt files are attached to this website. You should change the name of your input file and the name of f2amapping in the transaction editor (as shown in class).

Resource model

Background:

Resources are the processors, busses and other hardware things. We will define a hardware model with a bus. We will use a "Controller Area Network" or CAN bus.

Controller Area Network [5] is a communication bus developed by Bosch in the '90s. It is the "defacto standard" in cars, trucks and buses. The CAN-bus only defines the first 2 layers of the OSI-model (physical and data-link layer). Frames on a CAN-bus are of a size between 1 and 8 bytes. It uses a CSMA/BA (carrier sense multiple access with bitwise arbitration) arbitration mechanism. This mechanism ensures that no collisions can occur on the bus. When trying to transmit a frame on the bus, the controller first listens on the bus. If the bus is idle, the controller can start his transmission. Otherwise the controller waits until the bus is idle. When two or more controllers want to transmit at the same time, the bitwise arbitration mechanism ensures that the message with the highest priority is transmitted. The other must wait until the bus has returned to the idle state. It is an example of fixed priority scheduling mechanism. The CAN-bus can operate between 100 kbit/s and 1000 kbit/s. Every message on the bus has to have a unique ID!

Assignment:

Define a hardware system that consists of the following:

Front Left ECU	Sensor for wheel speed Brake Actuator	Mandatory
Front Right ECU	Sensor for wheel speed Brake Actuator	Mandatory
Back Left ECU	Sensor for wheel speed	Mandatory

[2] http://www.graphviz.org

	Brake Actuator	
Back Right ECU	Sensor for wheel speed Brake Actuator	Mandatory
ABSESP ECU	No sensors / actuators	Mandatory
YawAndSteer ECU	Yaw Sensor Steering Angle Sensor	Optional

All these ECUs are connected to a "Controller Area Network" with a speed of 100 kbit/s. You do not need to model the hardware ports for the sensors and actuators. The system all uses the same type of controller.

Deployment

Background:

Now that the software system and hardware system is defined, the hard part of integration (deployment) can begin. This consists of:

- Mapping of the software components to the hardware
- Mapping of the functions to tasks
- Mapping of application signals to frames on the communication bus
- Assigning priorities to tasks and messages

A bit of background on scheduling analysis:

Terminology:

Ri: Response Time of task i
Ci: computation Time of task i
Bi: Maximum blocking time (by something with a lower priority) of task i
Ti: Period or minimum time between arrivals of activation events
Ji: Jitter of task i

- Jospeph and Pandya (1986)

We start at the work of Joseph and Pandya [6] that is based on the seminal work of Liu and Layland in [7]. They state that the worst case response time of a task (or message) under a fixed priority scheduler can be calculated as follows:

$$Ri = Ci + Bi + \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{Ri}{Tj} \right\rceil . Cj \right)$$

hp(i) means all tasks with a higher priority than i.

You can see that all Ri is seen on both sides of the equality-sign. The equation says the following: The response time of a task i is his calculation time plus the maximum blocking time plus the times it gets interrupted by a higher priority task.

- Tindell et al. (1994): holistic schedulability analysis

Tindell extended the work of Joseph and Pandya in [8] to include jitters. These jitters come from the preemption mechanism of the RTOS or from interrupts from the arrival of messages. More important, he defined "holistic" schedulability analysis where not only one processor is used but multiple processors in combinations with a hardware bus. The technique updates the jitters of certain tasks when they must wait for a message arriving from the bus. This resulted in quite pessimistic behavior of the lower priority tasks. The mechanism was updated with static offsets. In [9], the system is composed of periodic transactions, each containing several tasks. Each task is released after some time, called the offset, elapsed since the arrival of the event that triggers the transaction. This is useful in those systems where task activations are timed precisely, at periodic intervals, to avoid the negative effects of jitter.

- Palencia (2002): dynamic offsets

Pallencia revised the work of Tindell in [10] to get a solution to the problems of task suspension and distributed systems if task offsets could be dynamic, i.e., if they could change from one activation to the next. For example, in distributed systems a task may be released when a previous task completes its execution and a message is received; this release time can vary from one period to the next.

We will use the technique of Palencia to check the worst-case response times of the tasks and messages of our ABS and ABS/ESP application.

Assignment:

Map the subcomponents of the software architecture to the defined resources. Don't do this in the ADL but in your report. The sensor/actuator components need to be mapped to the right ECU so that they can use the sensors/actuators. All other components can be freely distributed over the available ECUs. We will check whether the proposed configuration really does make its deadlines.

At this stage you should consider that mapping 2 interacting components to 2 different ECUs creates a delay due to the sending of the message on the bus (communication stack in software), the message being physically transmitted on the bus and the reception of the message on the receiver side. It also creates more memory overhead, since buffers need to be defined within the AUTOSAR communication stack.

Define in your report a software component to hardware mapping. And define the messages that need to be transmitted on the bus.

Finally we can define the tasks and messages. Since the ADL is not yet mature enough to really specify transactions, we will do this in another format. Download the timing estimate application from the axBench website. Open the example (below on this page) in the timing application. You can calculate the response time of the example application using the Actions->WCDOPS+ (worst case with dynamic offset) menu. Take a look at the gantt chart to see the worst-case end-to-end response times.

Open the file in a *.csv editor (like excel, …) and specify your own transaction of the ABS/ESP system.

Application requirements: The ABS and ESP both operate on a frequency of 50 Hz. The worst case end-to-end response time of both the ESP and ABS is 20 ms. We will verify whether our hardware can handle this in the ESP and ABS case. The source tasks are triggered by a Time Trigger (TT in the application). The others are triggered by the arrival of a message (PRED in the application). For the algorithm to correctly work, it can only use a single source. Define this source component with the highest priority and with a computation time of 0 and a trigger TT. The sensor tasks use this source task as there predecessor (PRED).

Define your tasks and messages, calculate the computation time needed and give the tasks and messages a priority. Put a deadline on the sink tasks. For each task assign it to the defined processor (processors are in the 100 range). For each message assign it to the CAN-bus (pid = 201).

Hint: Take a look at "list scheduling" techniques.

Runnable to Task rules:

You can map multiple runnables to a single task if they are mapped onto the same processor. But you'll need to define the execution order in AUTOSAR. If map multiple runnables to a single task, add in your report the order of execution of these runnables inside the task. If you do this the wrong way, you could end up using old data instead of new data.

Single task Ci calculation:

Add all the runnables WCETs to the task
For a task sending a message on the bus: add 0.41 ms (COM-stack + RTE)
For a task receiving a message from the bus: add 0.34 ms (COM-stack + RTE)
For a task sending a message to another task: add 0.005 ms (RTE)
For a task receiving a message from another task: add 0.006 ms (RTE)
For a task sending to a message in the same task: add 0.001 ms (RTE)
For a task receiving a message in the same task: add 0.001 ms (RTE)

CAN-bus message Ci calculation:

Time in ms = ((52+8*byte) + ceil((8*byte)/5))/busSpeed(kbit/s)
f.e. a 2 byte message will take maximal 0.72 ms on a 100 kbit/s bus.

Assigning a priority:

You should consider that a task (or message) with a higher priority has priority over tasks with a lower priority on the same processor.

Multiple input triggers:

Make sure that you trigger a task based on the predecessor with the largest WCRT. Otherwise it's possible that your runnables use old data.

The runnables WCET table:

ABS:

Runnable	**WCET [ms]**
Sensor_WheelSpeed_Read	1,2
VehicleSpeed_Calculation	2,88
Slip_Calculation	1,9
Slip_Compare	0,8
Brake_Actuator_Write	2,1

Extra ESP modules:

Runnable	**WCET [ms]**
Sensor_Yaw_Read	1,6
Sensor_Steer_Read	1,6
OverUnderSteer_Calculation	2,5
SelectBake_Calculation	6
MergeAbsEsp_Calculation	0,1

In your report:

- Define your resource and task mapping. Why did you do it like this?
- Copy the Gantt chart of the analysis.
- How did you schedule this? Did you use an incremental strategy? Why (not)?
- If your application is schedulable with your solution: why do you think this is? Do you think there is a better solution? How could we obtain this better solution?
- If not: why not and how could we get a schedulable solution? What are the parameters you can change to get a better solution?

References:

[1] FifthGear; "Bosch, ABS, TC, and VSC"; http://www.youtube.com/watch?v=1tSy5tHtT1g&feature=player_embedded#t=17

[2] Nenad Medvidovic and Richard N. Taylor, A Classification and Comparison Framework for Software Architecture Description Languages, IEEE Transactions on Software Engineering, vol 26, nr. 1, pp 70-93, 2002

[3] Cuenot, P. et al., Developing automotive products using the EAST-ADL2, and Autosar compliant architecture description language, Embedded Real-Time Software Conference, Toulouse, France, 2008

[4] AUTOSAR consortium, Technical Overview, STD, v.2.2.2, http://www.autosar.org/download/R3.1/AUTOSAR_TechnicalOverview.pdf

[5] Robert Bosch GmbH, CAN Specification, v2.0, 1991

[6] M. Joseph, P.Pandya, Finding Response times in a real-time system, The Computer Journal, Vol..29, nr.5, 1986.

[7] C.Liu, J.Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment", JACM 20(1), 1973, pp.40-61

[8] Tindell, K.,Clark, J., "Holistic schedulability analysis for distributed hard real-time systems", Microprocessing and microprogramming, Vol. 40,Nr. 2,pp 117--134,1994

[9] Tindell, K.,Clark, J., "Adding time-offsets to schedulability analysis", Technical report YCS221, University of York, 1994

[10] Palencia J.C., Harbour G. et al., "Schedulability analysis for tasks with static and dynamic offsets", Real-Time Systems Symposium, 2002