A Watchdog in Safety-Critical Java

Mikkel Todberg
Jeppe Lund Andersen

October 9, 2012

CONTENTS

CHAPTER

ONE

REQUIREMENTS

1.1 Problem

The Watchdog is based on the following problem setting. In a distributed system multiple modules communicate to accomplish a task as seen in Figure 1.1. In case of failure in one or more modules the system should take appropriate actions as other modules may be dependant on the failed module(s). The watchdog has the single goal of detecting such failures of any module and take appropriate action.

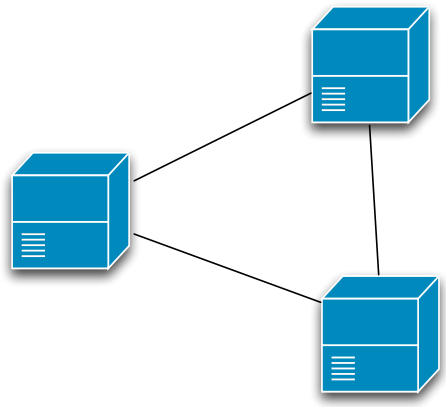

Figure 1.1: Three seperate modules working on a common task in a distributed setting

1.2 Environment Model and Setup

For the watchdog we set up a small hardware setup to model a scenario in which a module periodically provides sensor readings. As other modules would be dependant on these readings, the watchdog must detect any failures in the sensor module. In this case the watchdog only has to monitor a single monitor, however, the actual implementation should allow multiple devices to be monitored.

Figure 1.2 shows the setup of this small model. An Altera DE2-70 FPGA configured to run JOP will host the watchdog application implemented in Safety-Critical Java (SCJ). Connected to the Altera, is a Lego NXT UltrasSonic Sensor that provides

distance to objects measures. The connection between the watchdog and sensor happens on an I^2C bus. Each module on the bus is uniquely identified using a 7-bit address, thus 127 modules can be connected simultaneously. For analysis purposes, the watchdog will be limited to a maximum of 10 connected modules.

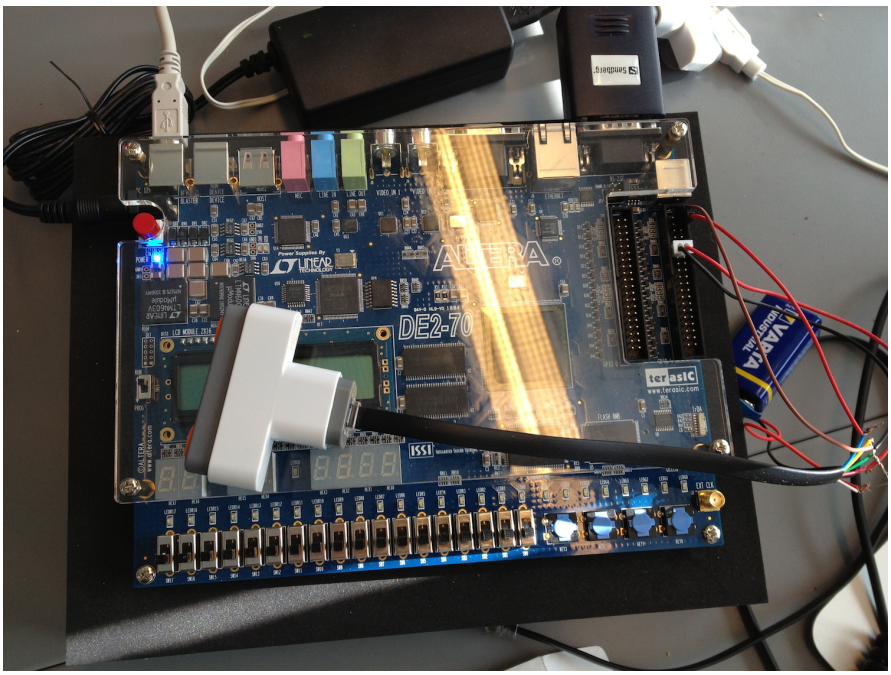

Figure 1.2: Altera DE2-70 and Lego NXT UltraSonic Sensor used in hardware setup

1.3 Tasks and Temporal Requirements

The tasks required for the watchdog depends on the communication flow. The watchdog can be designed to be the master that initiates and monitors each module, or each module can act as masters that regularly contacts and resets a timer in the watchdog. While the I^2C is a multi-master protocol, only one master and slave may communicate on the bus at a time. A multi-master approach in this distributed setting with modules contacting the watchdog, could lead to starvation of one or more modules due to repeated bus congestion, leading (incorrectly) to the watchdog assuming a module failure.

Instead we design the system around the watchdog acting as the sole master on the bus, in which failure of each module on the bus checked one by one. We designate three diffeerent tasks, (1) for handling I^2C communication with each slave module, (2) that checks status of each module based on the responses received in the first task and the final task (3) that handles a module failure. Table 1.3 shows the tasks along with related scheduling assignments.

The periods of the two periodic tasks are assigned such that the watchdog begins it monitor cycle every half second. We assign priorities according to deadline monotonic priority ordering. We argue that despite deadline enforcement is not available in SCJ it is suitable to consider deadlines due to the precedence relationship between the pinger and checker tasks - not only must both tasks be done executing

Task	Type	Period	Deadline	Priority
Pinger	Periodic	500	200	5
Checker	Periodic	500	100	10
FailureHandler	Aperiodic	-	50	15

Table 1.1: Task Set

Task	Type	Period	Deadline	Priority
Pinger	Periodic	500	200	10
Checker	Periodic	500	300	5
FailureHandler	Aperiodic	-	50	15

Table 1.2: Transformed Task Set

when either is to be released at a new period, the pinger task must run before the checker one. The deadline of the pinger is set on the assumption that each module times out 10 ms after the watchdog pings it, and in the worst case where the maximum of 10 modules times out, 100 ms will be spent on this alone. In order to correct the priorities and take the precedence relationship into account, we need the checker to have an offset. As in (Burns) we do this by stretching the deadline of the checker task and relate its deadline to the start of the transaction with the pinger, and not its own period and create a transformed task checker. Table 1.3 shows the (final) transformed task set.

CHAPTER

TWO

DESIGN AND IMPLEMENTATION

CHAPTER

THREE

WCET AND RESPONSE-TIME ANALYSIS