Automatic C code generation for Avionics Software based on VxWorks653 Platform

Ying Wang¹,²,a, Lei Wang²,b

¹College of informatics, Huazhong Agricultural University, China
²State Key Laboratory of Software Development Environment, Beihang University, China

a yingwang@mail.hzau.edu.cn, b wanglei@act.buaa.edu.cn

Keywords: Avionics Software, C code generation, VxWorks653, ARINC653, AADL.

Abstract. At present, model-driven development method has been gradually applied in safety-critical avionics software to improve reliability and productivity. Existing ARINC653 annex of the Architecture Analysis & Design Language (AADL) is used to model avionic software based on ARINC653-compatible OS, such as VxWorks653 commercial platform. In this paper, a refined C code generation algorithm is proposed to enable automatically generate VxWorks653Runnable code from the AADL ARINC653 models. Finally, a simplified autopilot system example is as a case study to show the validity of our approach.

Introduction

ARINC653 [1] is a new industry standard which provides a partitioned environment and application programming interface for avionics application. Modern avionic software usually runs on the ARINC653-compatible platform, such as VxWorks653 from Wind River [2], which is a famous commercial OS products conforming to this standard.

Currently, model-driven development method has gradually gained recognition to improve reliability and productivity for safety-critical avionic software, in both the academy and industry. Existing studies mainly focus on the code generation technology for general avionics platform [3], paying less attention to the ARINC653 partitioned platform. Only few research involves this, for instance, J.DELANGE et al. proposes a code generation strategy for partition system [4] based on the ARINC653 annex of the Architecture Analysis & Design Language (AADL) [5, 6]. But it is work in process and only considers simple code mapping strategy, lacking further refined code generation rules, such as complex communication behavior code of each ARINC process.

Therefore, we propose a refined C code generation approach for ARINC653 platform to automatically generate code runnable on the commercial VxWorks653 OS, from AADL models for ARINC653 [7]. The refined algorithm is respectively given in inter-process, intra-process and partition runtime level in details to describe concrete code generation process. We also implement a corresponding code generation tool to achieve automation. Finally, a simplified autopilot system example is modeled in AADL as input, and ultimate codes generated by our code generation tool are compiled and tested by the VxSim simulator. The experiment results shows the validity of our approach.

Code Generation Approach

Refined Code Generation. A refined code generation approach is proposed to generate ARINC653 partition initialization code, ARINC653 process communicationbehavior code and VxWorks653 xml configuration code from three levels as follows.
Inter-Process Level. The purpose of this level is to generate partition initialization code from the AADL ARINC653 inter-task communication models as shown in the slice of algorithm 1. For each partition represented by the AADL process instance (line 4), to instantiate all the ARINC process resource with attributes specified in the AADL models (line 5-6). Next, the inter-partition and intra-partition communication resource (e.g. destination sampling port and blackboard) are respectively generated by parsing the AADL port connections (line 7-10) and data subcomponents (line 11-13). Finally, the code generated in line 14-15 represents each ARINC process is started and the partition is activated by switching into NORMAL mode.

Algorithm 1: ARINC653 Partition Initialization Code Generation

1: Input: AADL ARINC653 Inter-Task Communication Model
2: Output: ARINC653 Partition Initialization C Code
3: begin
4: for all pImpl ∈ ProcessImplSet // create ARINC process resource
5: for all t ∈ ThreadSubSet(pImpl)
6: Output: CREATE_PROCESS(&processTable[id], &process_id, &ret_Code);
7: for all c ∈ DataPortConnecSet(p) // create inter-partition communication resource
8: t₁ ← srcComp(p); t₂ ← destComp(p); sp ← srcPort(p); dp ← destPort(p);
9: if (sp ∈ OutDataPort(t₁) && dp ∈ InDataPort(t₂)) then
10: Output: CREATE_SAMPLING_PORT("sp1", MAX_MESSAGE_SIZE ,
   DESTINATION, REFRESH_PERIOD, &sp_id, &ret_Code);
11: for all d ∈ DataSubSet(p) // create intra-partition communication resource
12: if (d.type == Blackboard) then
13: Output: CREATE_BLACKBOARD("board1", MAX_MESSAGE_SIZE ,
   &board_id, &ret_Code);
14: for all t ∈ ThreadSubSet(pImpl) // start ARINC process
15: Output: START(&process_id, , &ret_Code);
16: Output: SET_PARTITION_MODE(NORMAL, &ret_Code);
......

Intra-Process Level. In this level, it aims to generate communication behavior in each ARINC process from the AADL ARINC653 task behavior models described by the AADL behavior annex [8], as shown in the slice of algorithm 2. For each transition of an ARINC process (line 4-6), firstly, the inter-partition process communication code (e.g. READ_SAMPLING_MESSAGE) is generated according to the transition guard (e.g. an AADL in data port called sp) in line 7-8. Then, the intra-partition process communication code is generated according to the transition action (e.g. an AADL subprogram call called READ_BLACKBOARD) in line 9-10.

Algorithm 2: ARINC Process Communication Behavior Code Generation

1: Input: AADL ARINC653 Task Behavior Model
2: Output: ARINC Process Communication Behavior C Code
3: begin
4: for all tImpl ∈ ThreadImplSet // search for each task in AADL model
5: ba ← getBehaviorAnnex(tImpl);
6: for all tran ∈ transitionSet(ba) //search for each transition in AADL behavior annex
7: if (guard(tran)==sp? && sp ∈ InDataPort(tImpl)) then
8: Output: READ_SAMPLING_MESSAGE(sp_id, MESSAGE_ADDRESS,
LENGTH, VALID, &ret_Code);
9:      else  if  (action(tran) == READ_BLACKBOARD(rda, message)! && rda ∈ ReqDataAccessSet(tImpl)) then
10:        Output: READ_BLACKBOARD(ReqData(rda), INFINITE_TIME_VALUE,
              MESSAGE_ADDRESS, LENGTH, &ret_Code);

Partition-Runtime Level. Partition Runtime configuration code is generated according to the AADL ARINC653 partition runtime model in this level. It is composed of some configuration tables which provide static data areas accessed by VxWorks653 operating system to finish the initialization of the core module. Algorithm 3 shows partly the code generation process. For instance, some partition runtime configuration including inter-partition communication channel (line 4-8) and module schedule (line 9-13) are generated by respectively parsing AADL port connections, processors and virtual processors.

Algorithm 3: VxWorks653 XML Configuration Code Generation
1: Input: AADL Partition Runtime Model
2: Output: VxWorks653 XML Configuration Code Generation
3: begin
4:   for all sImpl ∈ SystemImplSet
5:      for all c ∈ DataPortConnecSet(s) // create inter-partition communication channel
6:         p1 ← scrComp(s);  p2 ← destComp(s);  sp ← scrPort(s);  dp ← destPort(s);
7:      if (sp ∈ OutDataPort(p1) && dp ∈ InDataPort(p2)) then
8:         Output: <Channel > element with <Source> and <Destination>
9:      for all pr← Processor(s); // create partition schedule window
10:         Output: < Module_Schedule> element
11:     Vp ← VirtualProcessorSet (pr);
12:    for all v ∈ Vp
13:       Output: < Partition_Schedule> under the < Module_Schedule>

Experiment and results

A simplified autopilot system example [9] modeled in AADL is shown in Fig 1, as be a case study of our code generation approach. The experiment environment is VxWorks653 OS from Wind River Platform. We first built the integration project by the Wind River workbench 2.3. Then the generated code is added to the project to be compiled and run by the VxSim, a VxWorks653 simulator. The running results and debugging information are partly shown in Fig 2. We can see that the resources in the flight management partition are created first. Then processes are set up and started. The results shows every process can communicate with other process correctly, which demonstrate the validity of our approach.
Conclusions

In this paper, we proposed a refined code generation approach for avionics software based on ARINC653, specifically VxWorks653 platform here, a commercial partitioned OS from Wind River Company. Detailed code generation algorithm is represented in three levels to respectively generate partition initialization code, task communication behavior code and partition runtime configuration code. A simplified autopilot system example codes generated by our code generation tool are compiled and run by VxSim simulator to show the validity of our approach.

Acknowledgment

This work is supported by the Research Fund for the Doctoral Program of HZAU under Grant No.52902-0900206153.

References
