Nios Code Generation Specification

Jonah Caplan

1 Introduction

The aim of this project is to develop an infrastructure for the automatic generation of C code for multicore Nios systems. We assume that an external model based design approach is used to generate control algorithms and that C code for these computations have already been generated in separate files (e.g. Simulink). The purpose of this tool is to efficiently map the control system onto an arbitrary platform while taking into account non-functional requirements such as deadlines, data flow, and criticality. The user must only specify the requirements for the system at a high level of abstraction and all intermdeiate code will be automatically generated.

This tool will not initially be geared towards solving codesign problems. We will assume a static hardware platform and ocnsider changing software requirements only. In order to facilitate later expansions, the tool will require the specification of the hardware in terms of generic model parameters. The task-mapping procedure will be platform independent and solve the problem generically even if we do not currently take advantage of this feature.

This document will provide the specification for the currently supported hardware models, the platform built from these components currently under study, the application models for analysis of software requirements, the mapping procedure combining both hardware and software models to generate a schedule, and the abstract template requirements for code generation.

2 Hardware Model

Hardware models are specified using an object-oriented semantics. The system is divided into hierarchical levels that are interpreted using static scoping rules to aid in the specification of larger models.

The cost in time of transmitting over communication channel models is omitted at this stage beyond specifying the connections between elements. Issues related to resource arbitration and interference are also not considered. The underlying infrastructure is assumed to be sufficiently quick and deterministic.

There are four categories of hardware models in the system: processor cores, peripheral cores, and memory and buses.

2.1 Processor cores

All processor cores are assumed to operate at the same clock frequency. Their parameters are determined during hardware design and are not altered during software mapping. They can be extracted from the *system.h* file generated from the .sopcinfo file by the Nios IDE during BSP generation.

Processor parameters are:

1. *Fault-tolerant*: While we do not have access to safety-critical Nios licenses, they do exist. We assume that a core can be designated as fault tolerant (FT) and that a cost is associated with fault tolerance (due to licensing, size, resource utilization, power consumption as appropriate for the scenario) and that it is therefore necessary to have fewer FT cores in the system.

2. *Scratchpad*: The processor must have a scratchpad in order to use fingerprinting as an error detection mechanism under our current implementation. The relevant parameters for the scratchpad will be listed separately.

3. *Timer*: The system timer will dictate the minimum period for events in the system. The timer period is statically assigned when specifying the hardware design in QSYS.

4. *Fingerprint Unit*: Fingerprint units will be necessary to allow the monitoring of processing cores (PC), i.e. those cores lacking FT capabilities, by a FT core.

5. *Data and instruction cache*: It may be necessary to disable data and/or instruction caches while executing critical tasks. It must be known if the processor is equipped with either.
6. *DMA*: A single channel DMA will be used to shuttle critical data in and out of the scratchpads.
7. *MPU*: An MPU will be required to ensure that each core is unable to maintain partitions between each core.
8. *Shared memory*: Shared memory space will be required to load instructions
9. *Interrupt signals*: The processor model must specify the actively connected interrupt signals.

This is a very general model of the processor. The mapping of tasks to cores will take place considering a very high level model of the system. The lower level parameters will only be used for code generation purposes. For the processor model, it is sufficient to consider whether or not each of these components are available. The details of each component will be hidden from the mapping problem at this higher level of abstraction.

2.2 Memory

Local scratchpads will be required for each core as well as shared main memory. A memory is defined simply as a start and end address. Memory latencies are not modelled. A memory model will also keep track of what other modules are connected to it.

There will be two partitioned sections of shared main memory. One to access common functions for redundant task executions and another for message passing between cores.

2.3 Peripherals

Certain details about the peripherals such as control registers will be entirely encapsulated in the template objects for the code generation phase. The higher level concerns that will dictate how these registers are sit will be included in the object representation of each peripheral. Mappings from the higher level concerns to code generation rules will be specified.

2.3.1 Timer

The frequency of the system level timer must be known in order to define the minimum resolution of time in the application model. Preexisting drivers exist and macros are generated by the Nios IDE to manage the timer. All relevant macros can be extracted from the generated code.

2.3.2 Fingerprint Unit

The fingerprint unit has the following features: maximum stack depth, statically set during hardware design, and block length size. The setting of

2.3.3 Memory Protection Unit

2.3.4 Comparator

2.3.5 DMA

2.3.6 μTLB

3 Application Model

3.1 Synchronous Languages - Building off the past

3.2 Code Rules

Any data passed between tasks must be done through pointers. Functions will return void pointers to predetermined data structures. These pointers will be passed between tasks. If tasks cannot share the same memory space, then the size of the data structure must also be known and the data will be copied to the appropriate location.