cfu-spec

Release 1.0.1

Jan Gray, and others

Jun 04, 2020

CONTENTS:

Draft 0.1 - WORK IN PROGRESS

CHAPTER

ONE

INTRODUCTION

In the coming winter of Moore's Law, computer system designers will employ application-specific custom hardware accelerators to boost performance and reduce energy.

A custom function unit (CFU) core is accelerator hardware that is tightly coupled into the pipeline of a CPU core, to add new custom function instructions that complement the CPU's standard functions (such as arithmetic/logic operations).

The present Composable Custom Function Unit (CFU) Specification aims to enable robust composition of independently authored, independently versioned CPU cores and CFU cores, and also of the application software libraries that use the CFU's new instructions. This enables a rich ecosystem of interoperable, app-optimized CFU cores and libraries, and straightforward development of app-optimized SoCs.

The specification comprises the software and hardware interfaces and formats needed for automatic composition of components into systems. From bottom up, these comprise:

- The CFU Logic Interface (CFU-LI) that defines the logic interface between a CPU core and a CFU core.
- The CFU Metadata Format (CFU-MD) that identifies the fixed and configurable properties and behaviors of CFUs and CPUs, enabling automated hardware and software composition tooling.
- The miscellaneous CFU Tools changes necessary for composition tooling.

CHAPTER

TWO

SCOPE

This specification has been developed under the auspices of the RISC-V FPGA Soft Processor Working Group but most of the CFU specification is ISA-neutral and may be applied to other configurable processors.

The initial CFU Spec focuses upon, and restricts its scope to, those function units that plug into the integer pipeline of a conventional microprocessor CPU, much like a two-operand, one-result fixed point arithmetic/logic unit (ALU).

For the purposes and scope of this Spec, a Custom Function Unit is defined as a hardware core that:

- accepts requests and produces responses, wherein:
  - requests may comprise a custom function ID and 0-3 integer request data words;
  - responses may comprise a success/error code and 0-2 integer response data words or an error ID;
- may be stateless, such that all function invocations are pure functions and are side-effect free, or may be stateful, with private internal state only. (See below.)

  Function units that directly access or update system memory, or architectural state of the CPU, including the CPU's command/status registers (CSRs), are out of scope.

CHAPTER

THREE

CONCEPTS: CUSTOM FUNCTION AND CUSTOM INTERFACES

A CFU implements one or more Custom Functions (CFs).

A Custom Function is a function from zero or more request data to zero or more response data.

A CFU may have state. A CF need not be a pure function. For example, invoking the same CF multiple times, with the same request data, may produce different response data.

CFU state, if any, must be private internal state. CFs must not read or write shared state.

> This rule afford simpler composition of CFUs.

CFs are bundled into Custom Interfaces (CIs).

CIs are identified by a unique integer CIID. Namespace management of CIIDs is TBD. (COM uses 128-bit GUIDs.)

CIs provide a namespace for CFs. A CF is identified by a (dense) integer CFID within a CI.

A Custom Interface is an immutable contract defining a set of CFs, the behavior of the CFs, and (if applicable) the necessary sequence of custom function invocations required for correct behavior of the interface.

Any organization may define a CI.

CIs are immutable. To change any aspect of the behavior of a CI, define a new CI.

> Implementers and clients of the original CI are not impacted.

Multiple CFUs may implement the same CI.

A CFU may implement multiple CIs.

> Custom Interfaces are modeled upon the Interface system of the Microsoft Component Object Model (COM), a proven regime for robust arms-length composition of independently authored, independently versioned software components, at scale, over decades.

CHAPTER

FOUR

CONCEPTS: COMPOSING AND COMPILING A CUSTOM SYSTEM

In the fullness of time, anticipating decades of customization by thousands of organizations, over thousands of applications, comprising tens of thousands of CFs, it may not be possible to carve up the limited opcode space of any target ISA into globally unique, fixed opcode assignments.

A Custom Function Unit Package comprises a CFU Core that implements the CFU-LI, packaged with CFU Metadata and its CFU Software (source code and/or binary library archive).

A system composition tool, TBD, compiles the application and libraries, the CPU and its CFUs, together into an SoC SW + HW design.

The CFU Software and Metadata identify which Custom Functions of which Custom Interfaces are used by the software. The tool uses this information to determine the (app-specific) custom target instruction set mapping required for the application to invoke the CFs.

The tool also generates the hardware shims necessary to interface the CPU(s) to the specific CFU(s), and to configure and CFUs (e.g. for operand width, or specific subset of CFs required) and perhaps the CPUs (operand latencies).

Each CI specifies some number of CFs: CI.NCF. The tool maps each CI's continuous range of CFs into a System CF index (SCFID) appropriate for that system and for the target ISA.

For example, if the app comprises two libraries, the first library uses functions 0,2,4 of CI-123 (with its CFs 0-4), and the second library uses functions 0,1,3 of CI-456 (with its CFs 0-3) the tool might establish the mapping

SCFID	CIID	CFID
0	CI-123	0
2	CI-123	2
4	CI-123	4
5	CI-456	0
6	CI-456	1
8	CI-456	3

or

SCFID	CIID	CFID
0	CI-123	0
2	CI-123	2
4	CI-123	4
8	CI-456	0
9	CI-456	1
11	CI-456	3

or

SCFID	CIID	CFID
0	CI-123	0
1	CI-123	2
2	CI-123	4
3	CI-456	0
4	CI-456	1
5	CI-456	3

or other encodings TBD.

The tool maps custom function invocations in the software to SCFID invocations in the compiled object code.

The resulting compiled and linked binary is hard-wired to this assignment of SCFIDs and therefore can only be run on the specific custom SoC.

> Yes, this is a significant shortcoming of this approach.

In one model, the CI-aware software is distributed in source and recompiled after the CFID to SCFID mapping is determined.

In another model, the CI-aware software may also be a compiled binary with relocations (fixups) to CI.CFID symbols. At link time the CI.CFID relocations are resolved to SCFIDs written to the correct fields of the CIs.

At execution time, a hardware CPU-CFU shim will map the SCFID invocation into an invocation of some CFU with some CI-scoped CFID.

For example, here if CFU Software uses CI-456.CF-1, the tool maps this into invocation of SCFID 6 in the compiled object code. Then at execution time, a hardware CPU-CFU shim maps SCFID 6 into an invocation of the CFU that implements CI-456, with CFID=1.

> As an alternative to the composition and opcode assignment system described above, it may be possible to define a single *custom instruction* instruction whose operands include not only the data operands but also a globally unique CIID.

4.1 CFU Logic Interface (CFU-LI) Specification

The CFU-LI defines a set of common hardware logic interfaces enabling robust (straightfoward, correct, stable, efficient) composition of separately authored CPUs and Custom Function Units.

4.1.1 CFU Logic Interface (CFU-LI) feature levels

The CFU-LI is stratified into four nested, increasingly flexible, increasingly complex feature levels. In each case, a CPU submits a CFU request and eventually receives a CFU respose. For each request there is exactly one response.

LI0: zero-latency combinational function unit: a CFU which accepts a request (CFID, request data) and produces a corresponding response (err/OK, response data) in the same cycle. Example: combinational bitcount (population count) unit.

LI1: fixed-latency, pipelined, in-order function unit: a CFU which accepts a request (CFID, request data, request ID) and produces a corresponding response (err/OK, response data, response ID) in a fixed number of cycles (may be zero). The request ID from the CPU is returned as the response ID from the CFU, and is used by the CPU to correlate the response to some prior request. Example: pipelined multiplier unit.

LI2: arbitrary-latency, possibly pipelined, in-order function unit, with request/response handshakes: a CFU which accepts a request (CFID, request data, request ID) and produces a corresponding response (err/OK, response data, response ID) in zero or more cycles. The request signaling has valid/ready handshake so that a CFU may signal it is

unable to accept a request this cycle, and response signaling has a valid/ready handshake to that a CPU may signal it is unable to accept a response this cycle. Example: one request-at-a-time, multi-cycle divide unit with early-out and response handshaking.

LI3: arbitrary-latency, possibly pipelined, possibly out-of-order function unit, with request/response handshakes: a CFU which accepts a request (CFID, request data, request ID, reorder ID) and produces a corresponding response (err/OK, response data, response ID) in zero or more cycles. The CFU may return responses to requests in a different order than the requests themselves were received. Example: a combined pipelined multiply and divide unit, wherein multiply requests are pipelined and require three cycles and divide requests are one-at-a-time and require 33 cycles, such that after accepting a divide request and then a multiply request, it provides the multiply response before the divide response.

To recap, each feature level introduces new parameters and ports to the CFU-LI, in particular:

- LI0: adds request (CFID, request data) and response (err/OK, response data).
- LI1: adds request ID/response ID correlation.
- LI2: adds valid/ready request and response handshakes.
- LI3: adds request reorder ID constraint, affording in-/out-of-order response control.

4.1.2 Rationale and use cases

This feature stratification keeps simple things simple and makes complex things possible. It anticipates and accomodates a diversity of CPUs that may be composed with CFUs. For example:

- a simple, one instruction at a time, CPU;
- a pipelined CPU;
- an in-order issue, concurrent execution pipelines, out-of-order completion CPU;
- a speculating, out-of-order issue CPU;
- a speculating, superscalar, out-of-order CPU;
- a hardware-multithreaded CPU that issues requests and receives responses for various interleaved threads; and
- a CPU cluster, of a multiple of the above types of CPU, that share a common CFU.

In general, a CPU that implements LI[k] can directly use CFUs that support LI[k], and can use CFUs for LI[i], i<k, by means of a CFU shim. For example,

- an LI1(LI0) shim implements LI[1] and encapsulates a LI[0] CFU by forwarding the CF request ID as the response ID.
- an LI2(LI1) shim implements LI[2] and encapsulates an LI[1] CFU by using pipeline clock enables and/or FIFOs to implement request and response handshaking. the response ID.

The use of request ID/response ID correlation also simplifies CFU sharing among multiple CPU masters. A CFU multiplexer shim may add additional source/destination routing data to a request ID so that subsequent response IDs directly indicate the specific CPU master destination.

4.1.3 TODO

TODO: discuss/decide if the above stratification is sufficient. For example, is it acceptable to bundle request handshake + response handshake into one feature level, or do we need a **lattice** "no handshake -> req handshake | resp handshake -> req + resp handshake"?

TODO: Is CIID always a metadata parameter, or in higher CFU-LI feature levels may it be provided dynamically on a port as part of the request port signal bundle? For example, suppose an LI[>=2] CFU core may implement >1 CI, but when the core is composed into a given SW + HW system by the system composition tool, it may be (must be?) configured and specialized to implement one fixed CI.

TODO: For CFUs with internal state (extra arguments, extra results, accumulators, etc.) define a standard name and behavior for state reset, state save/restore, interrogate state elements.

TODO: Decide how to name Custom Interfaces: VLNV, UUID, . . . ?

TODO: How much of the CFU-LI specification may be specified in IP XACT?

TODO: for LI>=2, is latency bounded? If so is max latency of the CFU (or individual CFID) provided to the CFU client at system composition time? Perhaps CFU metadata?