Hybrid Continuous-Discrete Computer: from ISA to Microarchitecture

Yipeng Huang, Columbia University
Simha Sethumadhavan, Columbia University

In this project, we design an instruction set architecture for a proposed hybrid continuous-discrete computer (HCDC) chip. The ISA harnesses the microarchitectural features and analog circuitry provided in the hardware. We describe the workloads that are suitable for the HCDC architecture. The underlying microarchitecture for the HCDC chip, including its controllers, datapaths, and interfaces to analog and digital functional units are specified in detail.

1. INTRODUCTION

We live in a world that is continuous: our eyes see scenes defined by continuous curves and intensities of colors, hear sounds formed by continuously varying pressure waves, and we sense velocity and temperature as continuous signals. In contrast, current computers operate in a discrete world, despite the continuous nature of the world at the scale our senses perceive. We finely divide time and space into quanta in order to model the real world on our digital computers.

The inherent costs of digital computing have been acceptable because of digital computers exponential growth in computing power relative to energy consumption and size. However, as scaling in digital computers comes to an end, we must explore alternatives to digital, discrete computing.

1.1. A Hybrid Discrete-Continuous Computer Architecture

An HCDC architecture that operates on continuous signals has been proposed [Sethumadhavan et al. 2012] for use in speeding up calculation of floating-point math, nonlinear math, and solving differential equations. Continuous computation promises to deliver adequate precision, low power, low latency, and low area cost accelerators for use in systems that extensively interact with the physical world.

The HCDC architecture is highly parallel [Figure 1], consisting of analog and digital functional units, which are full connected with a datapath, forming a single HCDC tile. An HCDC tile represents the smallest hardware design on which meaningful computation is possible. Multiple HCDC tiles may be aggregated, forming a HCDC fabric.

For the prototype HCDC system, we implement the functionality of a single HCDC tile, which is accessible by a prototype HCDC ISA. The ISA guides the HCDC tile through calibration, configuration, and control execution start and stop, providing control over the HCDC tile's analog and digital functional units and datapath.

While the prototype HCDC ISA exposes a lot of detail concerning configuration and calibration to the compiler and programmer, the production HCDC ISA will increasingly automate configuration and calibration of the HCDC tile, thereby concealing implementation details as part of the microarchitecture.

Where necessary in this document, we distinguish between the design choices for the prototype HCDC tile and for the production HCDC system.

For the production HCDC system, multiple HCDC tiles will be integrated onto a single HCDC fabric, which refers to a larger circuit design that consists of multiple HCDC tiles, an inter-tile datapath, and an inter-tile configuration and exception bus. That larger design will also be accessible by an extended HCDC ISA that supports use of multiple HCDC tiles.

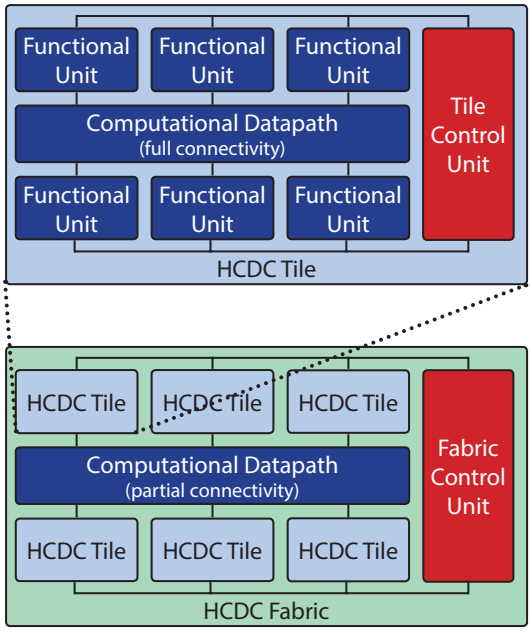

Fig. 1. HCDC system component hierarchy. The HCDC system consists of functional units organized as tiles. Multiple tiles are incorporated in a fabric.

2. HCDC WORKLOADS

Robotics controller software is a promising application domain for the proposed HCDC architecture. Prior work [Caselli et al. 1991; Liu et al. 2001], along with our own characterization, have found that robots expend a large portion of stored energy on computation. Furthermore, the hardware and software architecture for robotics are poorly matched to their unique demands.

Robotics computers often respond to analog inputs from their environment and react with motor control signals that are themselves analog. The inputs are often low precision and noisy, and accurate and stable behavior can only be guaranteed via system feedback loops that require low latency.

Using conventional digital embedded systems, robotics computers are designed as networks of hardware and software nodes that interact via an onboard network. The software for processing analog inputs exercise the floating point units in digital hardware, all of which are capable of calculation overly precise compared to imprecise inputs and outputs of the robots. As a result of these design choices, robots expend excessive power on network communication, on synchronization between computing nodes, and on excess floating point precision. This waste in power reduces the endurance of mobile robots.

In this report, we specify an HCDC ISA and matching microarchitecture that better matches the requirements of robotics workloads.

In future work, we will test the ISA by using it to support both robotics microbenchmarks and full robot software systems. The choice of microbenchmark kernels are informed by workload characterizations of robot software running on simulators. The kernels will consist of libraries from robotics software that consume a high proportion of the robotics runtime. These kernels include mapping, navigation, and differential equation solver libraries. The full robot software systems will enable performance evaluation of the HCDC architecture for a large cross section of robotics designs.

3. SYSTEM ARCHITECTURE

The HCDC system is a computer that is adept at solving differential equations and processing analog real world inputs.

3.1. Modes of Use

The HCDC system can be used as a standalone processor or in conjunction with a general purpose computer.

As a standalone processor, the HCDC system reacts to real world sensory inputs and directly controls motors or other physical devices. In this mode of use, the inputs and outputs of the HCDC system are both analog and continuous. A system master calibrates and configures the HCDC system for use, and the HCDC system then implements a black box function that operates with no intervention from the system master.

When used in conjunction with a general purpose computer, the system master calibrates and configures the HCDC system for use, and the HCDC system serves as a coprocessor for the main CPU. The CPU elects to offload computation that is more analog and continuous in nature to the more suitable HCDC system architecture.

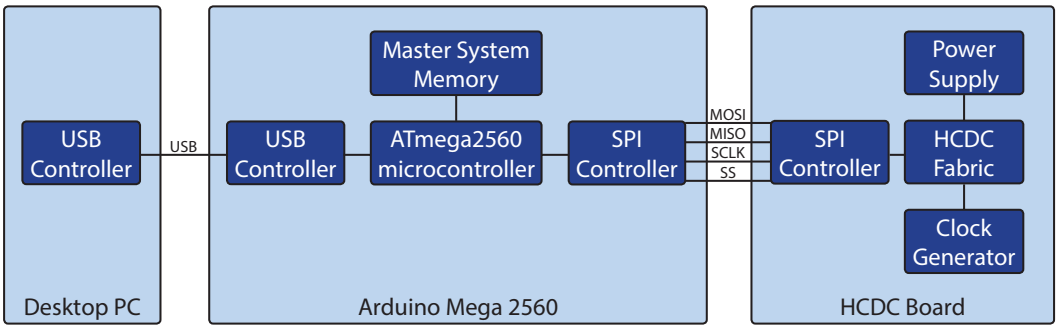

Fig. 2. Prototype HCDC system architecture.

3.2. System Microcontroller

The system microcontroller serves as an intermediary between the prototype HCDC chip and a general purpose computer—which can be a personal computer or a robotics controller.

For the prototype HCDC system, we use an Arduino ATMega2560 as the system microcontroller.

The Arduino Mega2560 shield features an ATMega2560 microcontroller, which operates at 16 MHz and has a 256 Kbyte memory. This is enough memory to hold up to 113 HCDC configurations and routers, along with testing programs.

For the production HCDC system, a custom controller and memory interface is preferred. As the HCDC system grows to include multiple HCDC tiles and HCDC fabrics, the program size will grow. The onboard memory size of the Arduino is limited, so a custom controller with a memory interface to a general purpose computer will be needed for the production HCDC system.

3.3. Host Interface

The system microcontroller connects to a general purpose computer using common peripheral interfaces. These include PCI, RS-232, and USB. We use USB because it is implemented on the Arduino Mega2560.

3.4. HCDC Fabric Interface

The system microcontroller connects to the HCDC chip via an interface that is included in the chip. Common microcontroller protocols include Serial Peripheral Interface (SPI) and I^2C. The Arduino Mega2560 implements both, and we choose to use SPI because it has better synchronization guarantees. The SPI pins are located at pins 50 through 53. The SS bar pin can be used select between multiple slave devices when multiple HCDC fabrics are present.

3.5. System Clock Generator

The HCDC system requires several different clocks:

3.5.1. clk_spi. The clock signal for the SPI controller operates at a maximum of 16 MHz, same as the ATMega2560 system clock. This clock synchronizes when bits are read and written on the MOSI and MISO pins of the SPI interface.

3.5.2. ctr_clk. The clock signal for the fabric and tile controllers operates at 24 time slower than clk_spi.

3.5.3. clk_scan. The clock signal for the testing the scan chain controller is not used in normal operation.

3.5.4. clk_ext. The clock signal for testing the LUT and DAC functional units is not used in normal operation.

3.6. System Power Supply

The HCDC fabric is connected to an external power supply that has to provide the following voltages and currents: ibiasadc, vtestadc, ibiasfan, vtestfan, ibiasdac, vtestdac, avdd, vcm, avss, ibiasint, vtestint, ibiasmul, vtestmul, esdvdd, esdvss, vdds, vdds. The specifications for what values these supplies need to be, and to what level of precision they must be generated, is still under design.

4. REGISTER SET

4.1. HCDC System Register Set

The addressing scheme for the prototype HCDC chip is specified in the address table spreadsheet.

4.2. HCDC Fabric Register Set

Fabric register set for production chip.

4.3. HCDC Tile Register Set

Tile register set for prototype chip. The register set includes registers for setting pointers and counters for configuration, data collection, data readout, and exception readout.

5. MEMORY ADDRESS SPACE

The addressing scheme for the prototype HCDC chip is specified in the address table spreadsheet.

6. INSTRUCTION SET ARCHITECTURE

This report is a specification for multiple layers of the continuous computing stack—from the ISA to the microarchitecture. The most desired findings are a first draft and viable ISA, along with the findings that would lead to designs for the HCDC programming language, library, and compiler. We recognize the workload suite and the hard-

ware design are evolving, and the division of responsibilities among the programmer, HCDC software, and HCDC hardware are likewise changing. The contents of this specification will change accordingly.

The instructions for controlling the production HCDC system should be parallel and quick, so that overhead time spent in establishing a circuit for analog computation is minimized, in order to serve workloads where analog and digital portions of code are finely interleaved. We should devote hardware and complexity to minimizing transition overhead.

For the first HCDC design, a lot will be exposed in the ISA. The prototype HCDC system exposes the details of functional unit layout and the datapath topology to the software stack, which includes the compiler, library, and programming language. As a result, these details become part of the architecture.

The ISA for the prototype HCDC system is expressed in the execution model [Section 11.3], state transition tables [Figure 3, Table VI], and the address tables, which describe the memory access scheme and register set for the prototype HCDC chip.

The execution model for the whole HCDC system is described in terms of the states of the system, along with the transitions and resulting instructions that are exchanged when the system transitions between states.

6.1. Endian Format

All values in the HCDC architecture are expressed in big endian format. This applies to configuration and exception protocol messages, along with the digital data I/O.

6.2. Transition MOSI Instruction Table

See Table I.

Table I. Transition MOSI Instruction Table

textbfMOSI Instruction	**Instruction Name**	**18 Bit Format**	**Effect**
Null	Null instruction	000,000,0000,00000000—write 0 to unused register	No operation
CfgStart	Configuration Start	000,000,0001,10000000—write 1 to Cfg register	Puts tile controller in CfgRecv state
CfgMsg	Configuration Message	[Section 10.2.3]	Sets configuration bits at functional units and crossbars
CfgStop	Configuration Stop	000,000,0001,00000000—write 0 to Cfg register	Puts tile controller in Idle state
ExecStart	Execution Start	000,000,0001,01x00000—write 1 to Exec register	Puts functional units and datapath in Exec state
ExecStop	Execution Stop	000,000,0001,00x00000—write 0 to Exec register	Puts functional units and datapath in Idle state
DataStart	Data Send Start	000,000,0001,0x100000—write 1 to Data register	Puts tile controller in DataS state
DataStop	Data Send Stop	000,000,0001,0x000000—write 0 to Data register	Puts tile controller in Idle state
ExpReq	Exception Request	000,000,0001,00010000—write 1 to Exp register	Puts tile controller in ExpI state
PowOff	Power Off	000,000,0001,00001000—write 1 to PowOff register	Puts tile controller in PowOff state

6.3. Transition MISO Response Table

See Table II.

Table II. Transition MISO Response Table

MISO Response	Response Name	18 Bit Format	Effect
Null	Null response	00,00000000,00000000—18 zeros	No operation
ExecDone	Execution done	10,xxxxxxxx,xxxxxxxx—ExecDone marker, and DataMsg	Signals to system master that HCDC fabric timed out
DataMsg	Data message	xx,ADC1data,ADC2data—data bytes from ADCs	Reports to system master ADC values
ExpMsg	Exception message	[Section 12.4.1]	Tile reports exceptions according to Tile Exception Protocol

7. CONTROLLERS

The tile controller and the functional units and Datapath have separate states.

7.1. Tile Controller States

See Table III.

Table III. Tile Controller States

Tile Controller State	Purpose	Control Signal State and Implementation
Power Up	Overhead time to let the power supply and clock generators stabilize.	The HCDC fabric enters this state once the power switch is flipped.
Idle	The tile controller is waiting for instructions from the system master.	All tile control signals are low.
Configuration Receive	The tile controller is accepting configuration messages, decoding the addresses, and writing configuration bytes to the functional units and Datapath. [Section 10.2.3]	The ctr_cfg signal is high. In the prototype HCDC fabric, the configuration messages are immediately used. In the production HCDC fabric, they would be stored in a buffer.
Exception Initial	The tile controller will report how many exception messages it will be sending. [Section 12.4.1]	The ctr_exp signal is high. In the prototype HCDC fabric, there will be eight exception messages, one each for each ADC and INT. In the production HCDC fabric, there would be a counter that tracks how many exceptions have been gathered.
Exception Send	The tile controller will send exception messages; the system master should expect as many exception messages as there had been agreed upon. [Section 12.4.1]	The ctr_exp signal is high. In the prototype HCDC fabric the exception messages will come directly from the exception bus. In the production HCDC fabric, they would be stored in a buffer.
Data Send	The tile controller will send the present value of both of the ADCs.	The functional units and Datapath can be either idle or executing. In the prototype HCDC fabric the data values will come directly from the ADCs. In the production HCDC fabric, they would be stored in a buffer.
Power Off	Execution stops and the tile waits to be shut down.	All tile control signals are low.

7.2. Functional Units and Datapath States

See Table IV.

Table IV. Functional Unit and Datapath States

Tile Functional Units and Datapath State	Purpose	Control Signal State and Implementation
Power Up	Overhead time to let the power supply and clock generators stabilize.	The HCDC fabric enters this state once the power switch is flipped.
Idle	The functional units and Datapath are idle.	The ctr_hold signal is high.
Configure	The functional units and Datapath are being configured. The LUT is available for writing. The decoder in the ADC microarchitecture is reset. [Section 10.2.3]	The ctr_cfg and ctr_hold signals are high.
Execution	The functional units and Datapath are in operation, doing calculation. [Section 11.3]	The ctr_start signal is high. The timing unit counts toward timeout. This state ends at timeout or at the instruction of the system master.
Exception	The functional units are having their exception messages read out. [Section 12.4.1]	The ctr_exp and ctr_hold signals are high.
Power Off	Execution stops and the tile waits to be shut down.	All tile control signals are low.

7.3. State Cross Products

The overall state of the prototype HCDC system is determined by both the state of the tile controller and by the state of the functional units and Datapath. Not all combinations are legal. Table V shows legal combinations of state.

Table V. State Cross Products

	Tile Controller State	Tile Functional Units and Datapath State
PowUp	Power Up	Power Up
Idle/Idle	Idle	Idle
CfgRecv/Cfg	Configuration Receive	Configure
Idle/Exec	Idle	Execute
DataS/Exec	Data Send	Execute
ExpI/Exp	Exception Initial	Exception
ExpS/Exp	Exception Send	Exception
DataS/Idle	Data Send	Idle
PowOff	Power Off	Power Off

7.4. State Transition Diagram

See Figure 3.

Some transitions are not shown. Refer to the State Transition Table below.

7.5. State Transition Table

See Figure VI.

8. DATA INPUT OUTPUT MODEL

9. CALIBRATION MODEL

The operation phases of an HCDC tile are calibration, configuration, execution, and exception reporting. After introducing the HCDC architecture, the following major sections will describe the calibration, configuration, execution, and exception models in sequence.

The procedure for calibrating functional units on an HCDC tile is evolving. This section describes a few design options for how to enable calibration.

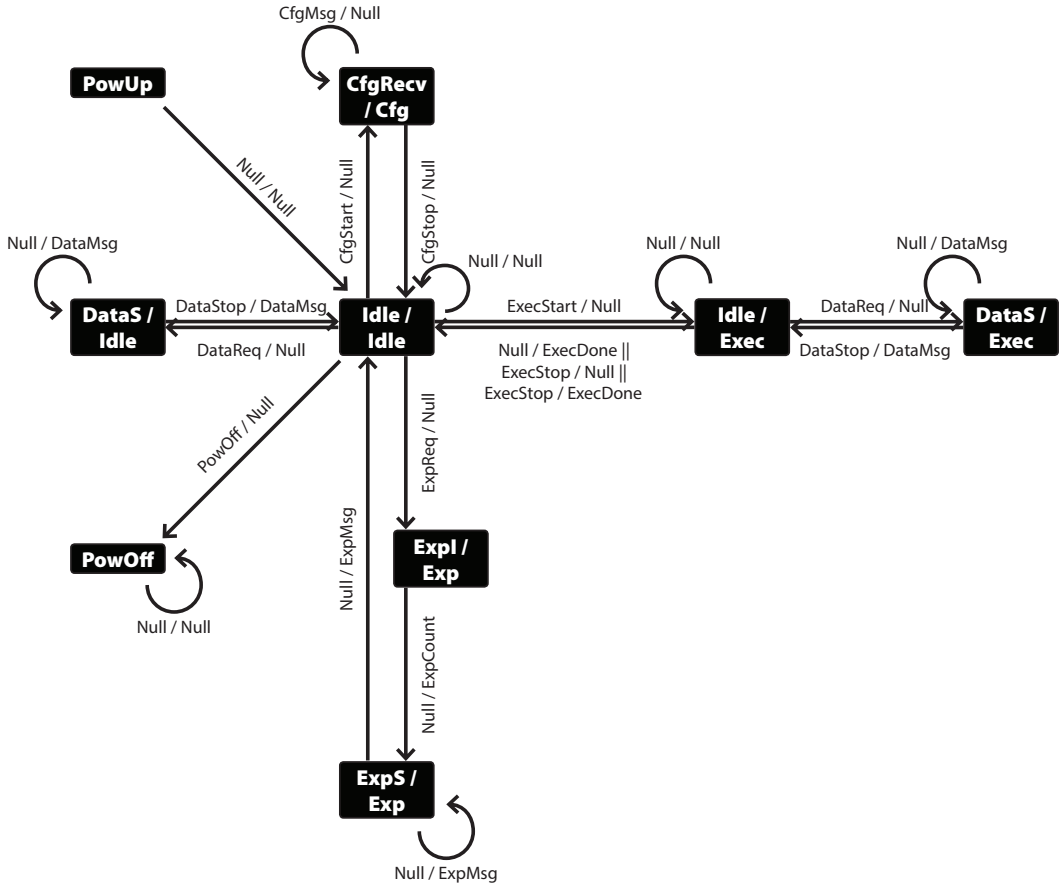

Fig. 3. State Transition Diagram.

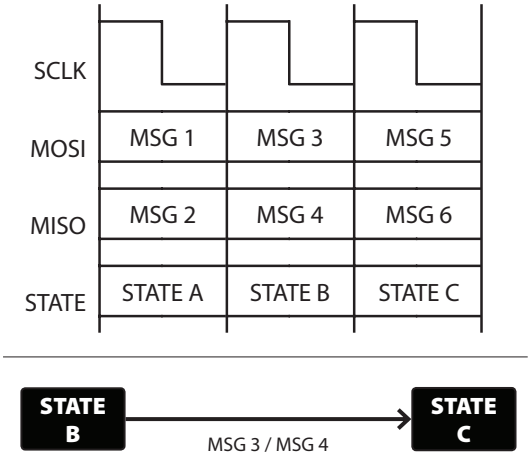

Fig. 4. State and Message Timing Diagram.

Table VI. State Transition Table

	PowUp	Idle / Idle	CfgRecv / Cfg	Idle / Exec	DataS / Exec	ExpI / Exp	ExpS / Exp	DataS / Idle	PowOff
PowUp		Null / Null							
Idle / Idle		Null / Null	CfgStart / Null	ExecStart / Null	DataStart / Null	ExpReq / Null			PowOff / Null
CfgRecv / Cfg		CfgStop / Null	CfgMsg / Null						PowOff / Null
Idle / Exec		Null / ExecDone —— ExecStop / Null —— ExecStop / ExecDone		Null / Null	DataStart / Null			DataStart / ExecDone	PowOff / Null —— PowOff / ExecDone
DataS / Exec		DataStop / ExecDone		DataStop / DataMsg	Null / DataMsg			ExecStop / DataMsg	PowOff / DataMsg —— PowOff / ExecDone
ExpI / Exp							Null / ExpCount		PowOff / ExpCount
ExpS / Exp		Null / ExpMsg					Null / ExpMsg		PowOff / ExpMsg
DataS / Idle		DataStop / DataMsg			ExecStart / DataMsg			Null / DataMsg	PowOff / DataMsg
PowOff									Null / Null

As an overview, each functional unit has a gain, and each input and output interface has an offset value. For a functional unit to deliver accurate results each gain must be calibrated to unity and each interface has to be calibrated to zero.

The MUL and FAN functional units have multiple copies of inputs and outputs, respectively, so each interface must be isolated and individually calibrated.

For the MUL functional unit, calibration is only effective if it can be provided an accurate unit value.

9.1. Calibration Design Options

We describe several options for implementing calibration.

9.1.1. Hardware Enabled Calibration. In hardware-enabled calibration, calibration requires no software intervention. Hardware routines prepare the circuit for calibration, exposing only a calibrate instruction to the ISA.

Per Functional Unit Calibration. In this design, functional units are as self sufficient as possible, exposing a nearly ideal I/O behavior to the datapath, independent of the HCDC tile configuration and operating conditions such as temperature. All functional units may be calibrated simultaneously as needed. This requires each functional unit to incorporate their own calibration unit.

However, this is a costly design option because it would inflate the area footprint of an HCDC tile. This is due to the difficulty in efficiently generating accurate bias currents.

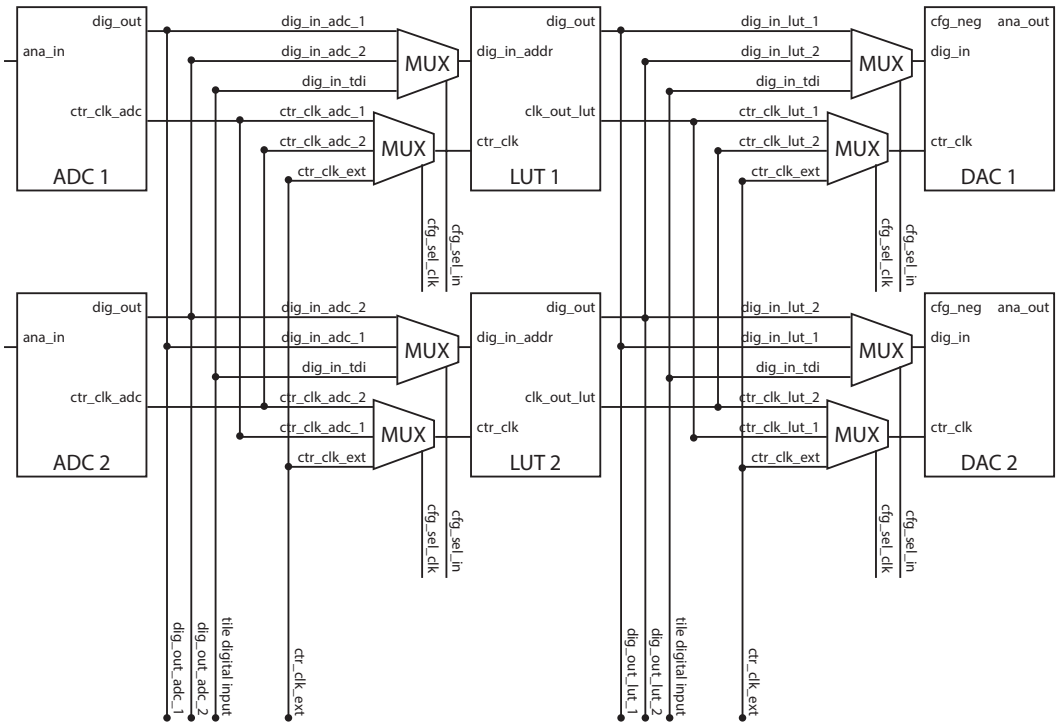

Fig. 5. Tile Digital Data Input Output.

Per Functional Unit Class Calibration. To conserve area, calibration units may be shared across multiple functional units. However, due to the distinct calibration needs of each class of functional unit, at best multiple functional units of the same type may share a common calibration unit.

This is the ideal design option for the production HCDC system since it offers the low latency of simultaneous calibration of multiple functional units, while at the same time reducing the footprint of an HCDC tile.

9.1.2. Software Enabled Calibration. In software enabled calibration, the fabric controller issues a microcode program that configures the datapath to expose a functional unit directly to outside the HCDC tile, one at a time. The HCDC tile would be instrumented to measure the input offset, gain, and output offset characteristics. In this design, the ADC should be calibrated first using manual instrumentation of the chip. After a reliable ADC is established, we can use the digital interfaces to the HCDC chip to calibrate the rest of the functional units.

Hardware Compensation. In hardware compensation, each functional unit contains registers that store compensation information on how recover from input offset, gain, and output offset variances.

An extension of this idea incorporates a table of registers that store the settings for multiple operating temperatures. The table may be reprogrammed intermittently to compensate for sources of error that are independent of temperature.

This requires additional area investment for the calibration memory at each functional unit, in addition to extra circuitry that uses the stored state to modify functional unit behavior.

Software Compensation. In software compensation, the functional units do not compensate for non-idealities in hardware. Instead, the configuration, gain, range, LUT key-value pairs have to be changed to get correct behavior from inaccurate hardware.

Software compensation can be used to correct for non-idealities that are not already compensated in hardware.

10. CONFIGURATION MODEL

While the datapath operates in analog, for the configuration bus and exception bus we leverage digital design.

10.1. Fabric Configuration Model

10.1.1. Fabric Controller Configuration.

FDIO Configuration Message. This section describes a special type of configuration handled by the fabric controller. The fabric controller unit recognizes a unique, special unused address as messages for the FDIO tile. The message sets multiplexing configuration for the fabric digital data I/O.

10.1.2. Fabric Crossbar Configuration. Fabric crossbars are not implemented in the prototype HCDC chip.

10.2. Tile Configuration Model

Tile configuration involves passing configuration messages to the functional units and crossbars, which then set up the datapath and parameters for performing a calculation.

In order to carry out configuration, the functional units and crossbars must first recognize that configuration messages are being passed. To signal this even during configuration, the configuration enable (ctr_cfg) signal is asserted true on the HCDC tile interface and to all underlying functional units.

Configuration for the datapath and for the functional units is then fed through the configuration message (cfg_msg) input. The protocol for configuring the HCDC tile is described in the address table spreadsheet.

The configuration protocol, which includes both bits for addressing the functional units and crossbars, is fed in to the tile via the configuration address and message (cfg_tile) interface. The address is decoded to row, column, and line selection signals. Once the correct line has been selected, the 16-bit configuration message is written to the line.

This microarchitecture isolates the digital decoder logic from the analog functional units and datapath. By keeping the digital control logic at the periphery of the tile, we are allowed to use abstractions provided by a digital design toolchain while keeping the logic separate from the sensitive analog circuitry.

10.2.1. Tile Controller Configuration.

10.2.2. Tile Crossbar Configuration. This section describes the protocol for configuring the analog datapath.

Crossbars receive switch configuration messages to configure eight switches on a vertical line.

Because datapath signals are in differential current mode, one functional unit output can only feed into one input, while multiple outputs can feed into one input. Therefore, within a vertical line of switches, only one switch can be set at a time.

10.2.3. Functional Unit Configuration.

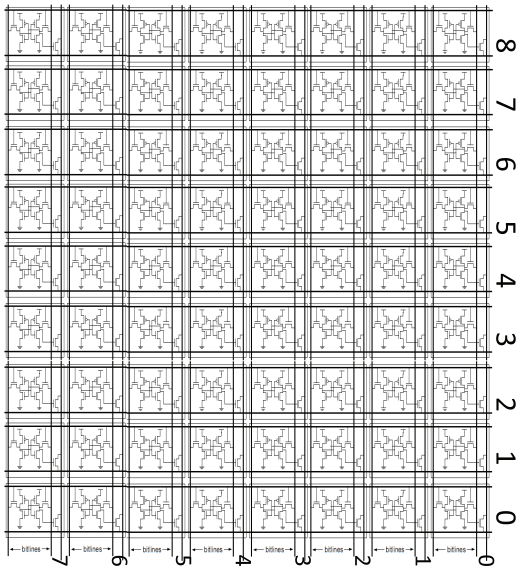

Fig. 6. Tile Crossbar.

11. EXECUTION MODEL

11.1. Tile Execution Start

Execution may start once all crossbars are configured and functional units have been calibrated and configured. Execution starts by asserting he ctr_start signal on the HCDC tile, which is a tile-wide control signal that routes to all INT functional units that signals beginning of integration.

11.2. Tile Execution Stop

Three stopping conditions are possible for the HCDC tile:

11.2.1. Indefinite operation. Useful when the HCDC tile is used as a standalone analog processor, or when the HCDC tile receives inputs from or sends outputs to other HCDC tiles as part of a larger HCDC fabric. Timeout length is set to zero signifying timeout is disabled. The ctr_stop signal is never asserted; the HCDC tile execution stops when the HCDC fabric issues a termination message.

11.2.2. Execution Timeout. Useful when the desired result is a signal value at a specific point in the simulated time. Timeout length is set to some interval of simulated time that corresponds to the physical time of the simulated computation. When the timeout is reached, the ctr_stop signal is fed to the INT functional units to hold the values still at that time.

11.3. Convergence

Useful when the desired result is a signal value that has settled to some tight envelope. The output of the computation may be the time it takes for any state variables to settle. To prevent indefinite operation in the case that the measured signal value does not converge, a timeout has to be set as a failsafe termination condition. This is not implemented on the prototype HCDC tile. This requires a convergence detection functional unit, which can be implemented with an envelope width detector followed

by a zero crossing detector. The proposed convergence detection functional unit asserts the ctr_stop signal once the solution converges.

12. EXCEPTION MODEL

Exception reporting is a significant feature of the HCDC tile and is a novel extension upon previous analog computer proposals. Accurate exception reporting greatly aids the compiler, programming language, and the programmer in debugging code. They are protection features in the HCDC system that help guarantee that functional units are used in a dynamic range that ensures precision and accuracy.

12.1. Exception Types

Overflow, underflow, and timeout exceptions are possible in the HCDC tile.

12.1.1. Overflow. Overflows occur when an integrator is charged beyond its capacity, or when the current on a line exceeds its configured range.

If a saturated integrator capacitor triggers the overflow exception, we do not distinguish between positive saturation and negative saturation because, in either case, the corrective course of action is to decrease gain AND/OR increase the range.

12.1.2. Underflow. This is a soft exception because the results in an underflow condition are not necessarily incorrect, just less precise (note, however, in chaotic or unstable computations an underflow likely leads to incorrect results).

If an integrators input line, output line, or integrator capacitor have less than 10% of their dynamic range used, the HCDC tile should throw an underflow exception on that functional unit. The corrective course of action is to increase gain and decrease the range: if the input buffer had been set to 0.1, it will be set to 1 instead; if the input buffer was already set to 1, then the previous output has to be amplified.

12.1.3. Timeout. Occurs when the tile controller times out before the simulated computation settles to a tight envelope.

12.2. Tile Exception Interfaces

During exception reporting, the exception enable (ctr_exp) signal is asserted true on the HCDC tile interface and to all underlying functional units. The functional units prepare exception messages, and the tile controller can then read the exception messages in a pipelined fashion through the exception message (exp_message) interface. The protocol for communicating exceptions in the HCDC tile is described in the section Tile Exception Protocol.

12.3. Tile Exception Microarchitecture

The exception bus is a scan chain that visits all flip-flops that contain exception signals.

12.4. Tile Exception Protocol

Tile exception protocol messages are 16 bits wide.

12.4.1. Functional Unit Exception Message. A functional unit exception message describes all the exceptions that have occurred in a tile. The HCDC tile needs as many cycles to read out the functional unit exception messages as there are number of functional units that generate exceptions.

Acknowledgments

ACKNOWLEDGMENTS

The authors would like to acknowledge the work of the HCDC team members Ning Guo, Sharvil Patil, Tao Mai, and Chi Cao of the Columbia Integrated Systems Lab. This work is realized with the guidance of professors Yannis Tsividis and Mingoo Seok.

REFERENCES

CASELLI, S., FALDELLA, E., AND ZANICHELLI, F. 1991. Performance evaluation of processor architectures for robotics. In *CompEuro '91. Advanced Computer Technology, Reliable Systems and Applications. 5th Annual European Computer Conference. Proceedings.* 667–671.

LIU, J., CHOU, P., BAGHERZADEH, N., AND KURDAHI, F. 2001. Power-aware scheduling under timing constraints for mission-critical embedded systems. In *Design Automation Conference, 2001. Proceedings*. 840–845.

SETHUMADHAVAN, S., ROBERTS, R., AND TSIVIDIS, Y. P. 2012. A case for hybrid discrete-continuous architectures. *Computer Architecture Letters 11,* 1, 1–4.