Cognitive Sensor Platform

Mark Mc Dermott
Department of Electrical and Computer Engineering, University of Texas at Austin

Article Info

Article history:

Received Jan 21, 2014
Revised May 31 2014
Accepted Jun 17, 2014

Keyword:

Cognitive Sensor Platform
Composable Systems
Dataflow Processor
Synchronous Dataflow

ABSTRACT

This paper describes a platform that is used to build embedded sensor systems for low energy implantable applications. One of the key characteristics of the platform is the ability to reason about the environment and cognitivelymodify the operational parameters of the system. Additionally the platform provides to ability to compose application specific sensor systems using a novel computational element that directly supports a synchronous-dataflow (SDF) programming paradigm.

Corresponding Author:

Mark Mc Dermott
Department of Electrical and Computer Engineering,
University of Texas at Austin
1 University Station C0803, Austin, Texas 78712-0214
Email: mcdermot@ece.utexas.edu

1. INTRODUCTION

The next step in the evolution of intelligent sensors is towards cognitive sensors. Cognitive sensor platforms have the capability to reason about their internal/external environmental conditions and then modify their system behavior. The result is a sensor platform that dynamically optimizes its operation to meet the system metrics. The hierarchy of advanced sensor systems is generally described as having three distinct levels of capabilities. This classification is graphically illustrated in Figure 1.

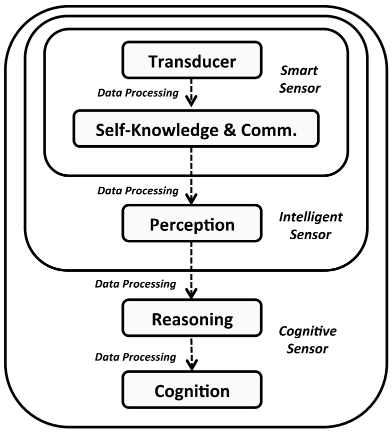


Figure 1. Computational hierarchy of advanced sensor systems [1]

The first level is a “smart” sensor system where the sensor is able to identify its purpose and can communicate information to and from other devices. The second level is generally classified as “intelligent” and this is achieved by adding the ability to recognize, interpret and understand sensor stimuli. The third level of capability is the cognitive sensor that adds reasoning and cognition to the intelligent sensor system, allowing it to make decisions based on the sensor stimuli and to be aware of the environment that the sensor is operating in [1].

Each level of hierarchical capability requires a corresponding improvement in computational and energy performance. While transistor scaling has provided some of the additional computational efficiency, the energy supply for embedded sensors is staying virtually constant since Moore’s Law does not apply to battery technology. That said, Koomey, et al, observe that computational-efficiency (measured in Computations/Joule) is improving at a similar rate to Moore’s Law [2]. This may be adequate for workloads whose computational efficiency requirements remain constant from one generation to the next however for next generation sensor workloadscomputational-efficiency is not improving at a fast enough rate if transistor scaling is the only source of improvement. From a system design perspective, optimal computational-efficiency is achieved by “impedance matching” the four domains that comprise a sensor design. The four domains include the algorithmic domain, the software program domain, the hardware micro-architecture, and lastly the silicon technology. This is described by the following pseudo-equation:

$$\text{Computational} - \text{Efficiency} \propto \frac{\#\text{Inst}}{\text{Task}} * \frac{\text{TP}}{\#\text{Inst}} * \frac{\text{PP}}{\text{TP}} * \frac{\text{LL}}{\text{PP}} * \frac{\text{ns}}{\text{LL}} * \text{Joules} \quad (1)$$

where:

#Inst = number of executed instructions for a Task
TP = Instruction Trace Parallelism
PP = Processor Parallelism
LL = Levels of Logic
ns = nanoseconds or 1/frequency

The number of instructions per task is the mapping of the algorithmic domain by the software compiler into single or multiple software threads. Ideally the number of software threads is matched to the number of processing elements or hardware threads. The processing element(s) would be designed to provide the optimal energy-performance in order to accomplish the task as determined by levels of logic needed per clock cycle. It can be seen that solving the equation “as is” results in a value of “1 Joules-ns/Task”, which indicates anideal impedance match between all components of the equation. If there is a mismatch between any of the domains the computational efficiency of the machine will be non-optimal.

It is generally accepted that general-purpose computing platforms do not have the energy-performance characteristics needed for low energy sensor systems. Hard-coded logic would provide the most optimal computational-efficiency but at the expense of reprogrammability. The computational element used in this platform is implemented using an energy efficient programmable microcoded engine with a single cycle datapath. The levels of logic and the ratio of control logic to datapath logic are optimized for low energy applications.

The software programming paradigm is also optimized for sensor applications. Sensor systems are reactive and can be implemented using a number of different reactive programming models including Dynamic Dataflow (DDF), Synchronous Dataflow (SDF), Discrete Events (DE), Petri nets and Khan Process Networks (KPN) [3]. This platform uses Synchronous Dataflow (SDF) as it is a special case of dataflow modeling where the flow of control is predictable at compilation time [4]. The microcoded computational element used on this platform is designed to directly support the SDF event-driven programming paradigm.

2. REQUIREMENTS FOR A COGNITIVE SENSOR PLATFORM (CSP)

This platform is designed be used in deeply embedded applications such as medical implants, structural implants and remote sensing. The key figure of merit for this class of embedded sensors is energy-performance/volume where battery volume is the limiting factor as it determines the number of joules available for system operation. The addition of cognitive capabilities is necessary for these types of unattended applications as it is generally not feasible to routinely replace the battery or sensor(s) in these applications. Cognition in the context of a sensor platform is defined as the “process of knowing, including aspects of awareness, perception, reasoning, and judgment” [1]. Figure 2 shows conceptually how the process of knowing applies to a cognitive sensor system.

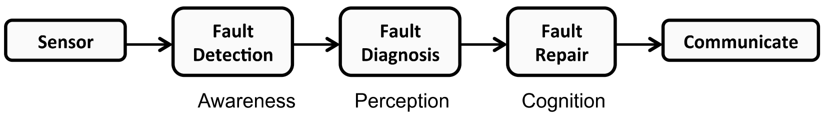


Figure 2. “Process of Knowing” in a cognitive sensor system

This “process of knowing” drives the following baseline functional requirements of the CSP. These include the ability to:

- Perform self-diagnostics and self-calibration.
- Compensate for systematic errors, system drift and random errors produced by system parametric changes such as sensor aging, battery aging.
- Fuse data from homogeneous and heterogeneous sensors.
- Detect and repair corrupted data
- Reason about the state of the system and perform needed services to maintain optimal system performance.
- Anticipate potential systematic changes and modify operational behavior.
- Transmit/receive information to/from other devices.

The CSP functionality requirements described above drives a number of key architectural features including:

- Debug and diagnosis
- Time Stamping
- Adaptive capabilities including:
  - Configurable data lookup capability
  - Reconfigurable event-driven programming
  - Dynamic sampling and frequency scaling
  - Dynamic data precision
- Fuzzy logic capability
- Data fusion capability

2.1 Debug and Diagnostic Capability

The CSP provides a computational (digital) diagnostic mode that utilizes auxiliary channels to confirm that the primary channels are performing as expected. Injecting calibration tokens into the SDF network and analyzing the response to confirm computational accuracy accomplishes the diagnostic. The source of the calibration tokens is controlled by the Channel Nodes in response to a control signal from the Debug Unit. A wide range of diagnostics can be accomplished using “digital” tokens. The discrete value and temporal response of the SDF network can be analyzed by the DPE utilizing this diagnostic mode. In addition to the “digital” diagnostics, the Debug Unit can inject “analog” values into the sensor readout circuitry. The values are generally limited to those that can be easily implemented using voltage references that are insensitive to aging, power supply variations and device variations.

Calibration of the sensor readout circuitry is accomplished using a programmable content-addressable lookup table (CLT) in the sensor data-conditioning element. The CLT is initialized at reset with the stored calibration data from an external memory. During system operation, the CLT values can be updated to adapt to environmental changes in the sensor transducer. The modified CLT data can be transferred to the external memory so that subsequent reset operations load the new calibration data into the CLT.

2.2 Time Stamping

The time stamp function uses a timing reference to generate a unique value that is tagged to the token data produced by A/D converter. This timing reference can be generated internally in the CSP or derived from an external network synchronization signal [5]. If the timing reference is derived internally the external receiver must synchronize to the internal timing reference by algorithmic means [6] [7]. The number of samples is determined by the operation that the CSP is intended to perform. For example if five samples are being averaged to a single datum then the timing window is valid for five samples and a single time stamp value is issued. The time stamp value is incremented for every sample and one of the five time stamp values is used to tag the data depending on the algorithm being performed. In this example the third time stamp value could be used when averaging five samples. This technique provides a built in time reference for all data that is being processed by CSP. The time stamp value can be transmitted along the output token data via the communications element for external processing.

2.3 Adaptive Capabilities

The CSP detects environmental changes by tracking factors such as the rate of change of sensor data, computational data errors, battery voltage, temperature, etc. The CSP can adapt to these changes using pre-defined rules. One of the most useful adaptation techniques utilizes fuzzy logic decision-making [8]. This is described below in the next section. The CSP has a content address lookup table (CLT) that can be dynamically reprogrammed during system operation. The CLT is used to calibrate sensor data, provide complex logic functions and for fuzzy logic operations such as the defuzzify step where the antecedents are mapped to deterministic output values.

The CSP can dynamically reconfigure its program sequencing, which is required for cognitive systems that need to dynamically modify their algorithms based on operational conditions [9] [10]. This capability in the CSP is achieved by using an actor/event queue. The relative order of the how the operations are processed can be dynamically changed by reentering or reordering the actors in the queue. The actors can be inserted into the queue in either FIFO or LIFO order. The actors are executed from the “Top-of-Stack” as shown below in Figure 3. Each operation is mapped to a microcode routine that terminates using a “Wait-for-Event” instruction. Interrupts and other asynchronous events can also enter operations to the queue by inserting them into the execution stream. These events are squashed from the actor-queue after they are executed. The actor-queue is circular which allows it to execute continuously until a break condition is encountered. Typically a break condition occurs when new tokens data are needed.

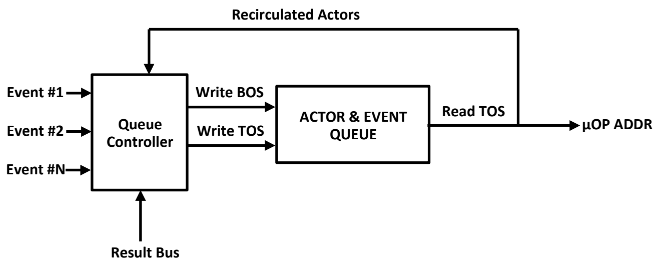


Figure 3. Actor and Event Queue

The Debug Unit preloads the actor-queue with a prescribed sequence of operations via the JTAG scan chain. As the CSP becomes operationally and conditionally aware of its environment the operation-queue can be saved to external memory so that the new state can be reloaded during the next reset cycle.

The datapath in the CSP is designed to use signed saturating arithmetic. The data precision can be dynamically modified to save power by changing the saturation limits and scaling the data tokens as needed. Additionally the CSP can modify the sampling rate of the sensor data if the rate of change of the incoming data is low.

2.4 Fuzzy Logic Capability

The CSP contains a fuzzy logic engine to analyze sensor data and make systematic adjustments to the operation of the platform plus provide specialized functions like data fusing (described in the next section). This engine is implemented using a combination of specialized hardware functions and microcode routines. The specialized hardware consists of logic to perform minimum, maximum and table lookup functions. The microcode engine performs the membership, rule evaluation and weighted average functions [11].

As mentioned above the CSP can control energy usage by controlling the sampling rate of the sensor data and controlling the clock frequency. A fuzzy logic based algorithm is used to determine the sampling rate by analyzing the change and the rate of change of the input data. Figure 4 below shows the flow diagram for the fuzzy logic engine. The first step performs a membership evaluation on the inputs. There are two inputs; one is the absolute value of the sensor data change and the second input being the rate of change of data change. The second step performs the evaluation of the rules that determine the energy levels. The third step converts the energy levels to control signals to drive the sampling rate and/or the clock frequency of the CSP.

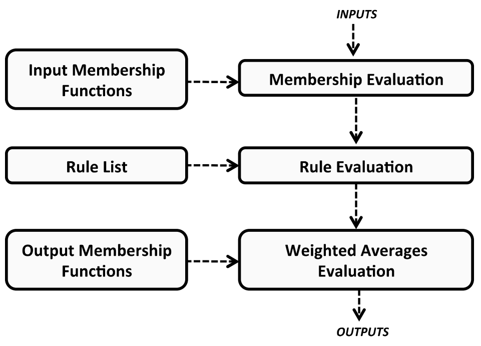
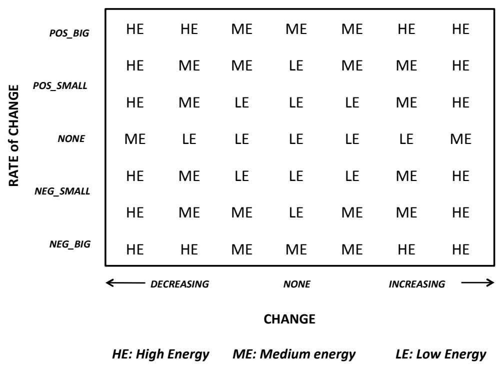

INPUTS
Input Membership Functions
Membership Evaluation
Rule List
Rule Evaluation
Output Membership Functions
Weighted Averages Evaluation
OUTPUTS

Figure 4. Fuzzy logic flow diagram

Figure 5 below shows a table of the outputs from the rule evaluation of the energy function. The lower energy operations occur when the input data is not changing very rapidly and the rate of change of the change is also not changing very rapidly. Conversely if the input data is changing rapidly a higher energy is required to process the data.



HE	HE	ME	ME	ME	HE	HE
HE	ME	ME	LE	ME	ME	HE
HE	ME	LE	LE	LE	ME	HE
ME	LE	LE	LE	LE	LE	ME
HE	ME	LE	LE	LE	ME	HE
HE	ME	ME	LE	ME	ME	HE
HE	HE	ME	ME	ME	HE	HE

RATE of CHANGE
POS_BIG
POS_SMALL
NONE
NEG_SMALL
NEG_BIG
DECREASING
NONE
INCREASING
CHANGE
HE: High Energy
ME: Medium energy
LE: Low Energy

Figure 5. Energy usage rule evaluation table

2.5 Data Fusion Capability

Data fusion is important for data reliability and robustness, data compression operations and for composing complimentary or spectral data [12]. The CSP supports low-energy data fusion using a combination of microcode routines and the fuzzy logic engine to perform the data fusion operations [13] [14]. This low-energy approach is preferred over mathematically intensive algorithms using least square-based estimation methods such as Kalman Filtering [15] or probabilistic methods such as Bayesian analysis [16]. For those sensor applications where a more accurate data fusion algorithm is needed a hybrid of Kalman filters and fuzzy logic can be implemented with minimal additional logic [17]. The limitation of this hybrid approach is the data precision provided by the DPE and the increased energy usage.

Data from either single or multiple sensors can be fused into a composite data stream. The fused data contains more information than the original inputs and is used either locally in the CSP and/or transmitted to a receiving node for further processing. The fuzzy data fusion algorithm involves aggregating data from the input sensors and utilizing predictive data from past aggregation to generate fused data and optional sideband data as shown below in Figure 6.

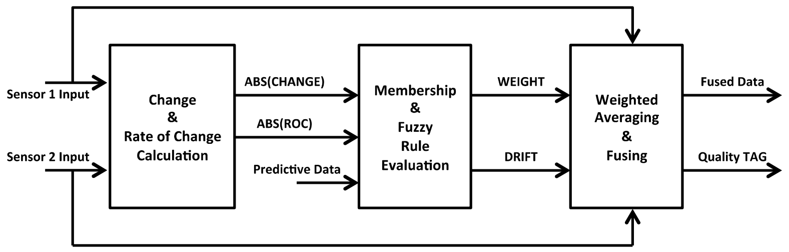

Sensor 1 Input
Sensor 2 Input
Change & Rate of Change Calculation
ABS(CHANGE)
ABS(ROC)
Predictive Data
Membership & Fuzzy Rule Evaluation
WEIGHT
DRIFT
Weighted Averaging & Fusing
Fused Data
Quality TAG

Figure 6. Fuzzy data fusion

In this example, the data from two asymmetrical sensors are fused together to produce a composite signal that has the key characteristics from each sensor. The first operation involves determining the absolute value of the change and rate of change (ROC) for the two sensors. The second operation performs a membership evaluation and series of fuzzy rule evaluations to produce a weighting factor and a sensor drift value. The third operation uses the weighting and sensor drift information to produce the fused data and a quality tag. The quality tag is sideband data that can be used by the CSP to adapt to sensor drift, data sampling issues, etc.

3. CSP ARCHITECTURE

The CSP is an event driven synchronous data flow architecture. It is “composed” by instantiating functional elements that are connected via channels. The functional elements provide key operational services commonly called “actors” in a dataflow system. In the current implementation the channels are modeled as bounded FIFOs. The information datum that is communicated via the channel interface is referred to as a “token”. Figure 7 below shows the five basic functional elements that are used to compose a CSP:

1. Sensor Element - Transducer and Readout Circuitry
2. Sensor Data Conditioning (SDC) Element
3. Dataflow Processing Element (DPE)
4. Communications Element
5. Debug Element

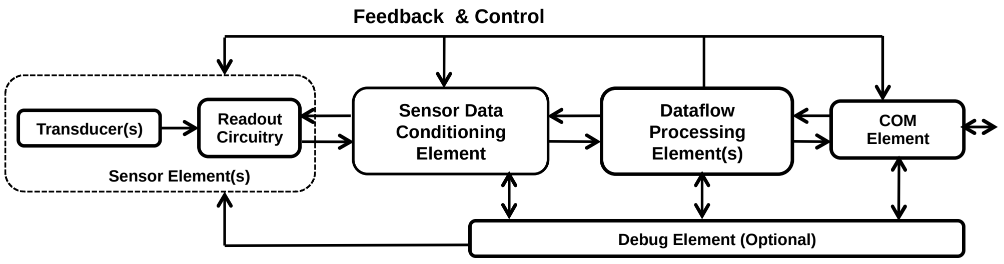


Figure 7. CSP Block Diagram

The output from the readout circuitry in the Sensor Element will generally be an analog signal that will require some additional analog processing such as filtering, amplification and conversion to a digital representation using an analog-to-digital converter (ADC). This additional processing is done in a preprocessing unit in the SDC element. The CSP may optionally contain one or more dataflow-processing elements (DPE) that process the data from the SDC and communicate the output data via the COM element to a receiving device. In addition to these four elements, an optional debug element can be used to debug functional failures and reconfigure the CSP during normal operations.

This platform is designed to support a reasonably wide variety of sensing techniques including, voltage, resistive, capacitive, inductive, optical, magnetic, force and acceleration. Typical embedded sensors would include strain gauges, piezoelectric devices, phototransistors, hall-effect devices, thermo-couples, ion-sensitive transistors, capacitive displacement devices, and bio-sensing devices [18].

The sensor data-conditioning element provides a broad range of data conditioning and transformation services. These services include signal conditioning, signal conversion, detection functions, data-reduction and data-fusion.The high-level block diagram for the SDC element is shown below in Figure 8. The SDC contains three basic units:

1. Preprocessing unit (PPU) - includes filters, A/D converters, etc.
2. Functional services unit (FSU) - performs data conditioning services.
3. Channel node(s)

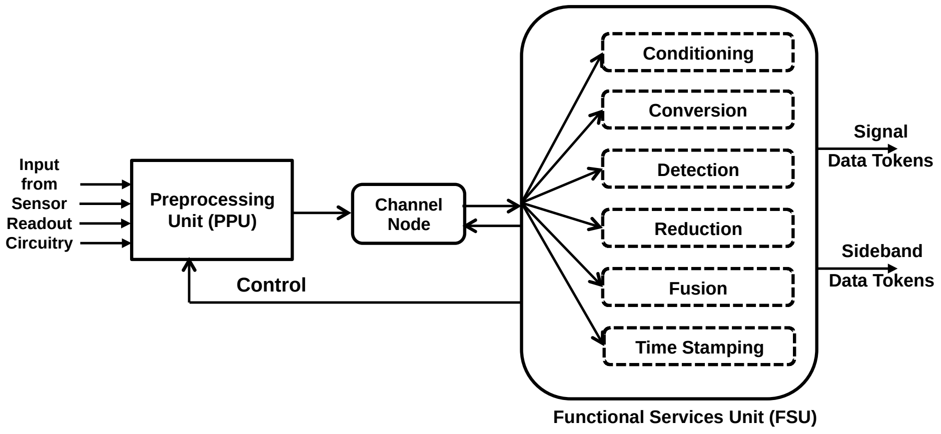


Figure 8. Sensor Data Conditioning (SDC) Element Block Diagram

A typical preprocessing unit contains some combination of the following components: filters, amplifiers, analog-to-digital converters (ADC), sample-hold circuits and analog multiplexors as shown below in Figure 9.

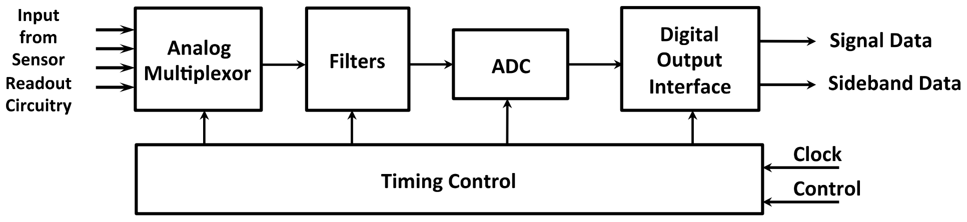


Figure 9. Typical configuration of a PPU

The FSU can be implemented using a synthesized hard-coded logic unit or with a microcoded engine such as the DPE. The synthesized implementation is preferred for basic services including averaging, data compression, transition counting, event triggering and threshold detection. The microcoded engine is best suited for the complex services, as they typically require complicated processing of temporal data. These include services such as linearization and smoothing, edge detection, data suppression, data fusion, filtering and signal reproduction. Channel nodes handle all transmission and buffering of data between the Functional Service Unit (FSU) and the PPU. The FSU is “fired” when the channel node has buffered all the token data from the PPU and is ready to transmit it. Channel nodes are also used to transmit data between multiple dataflow processing elements as described below in Section 4.

The dataflow processing element (DPE) used for this platform is implemented as a stack-based microcoded engine with advance features such as nested looping, conditional execution, repeat execution and a programmable lookup table for reconfigurable functional operations [19]. Figure 10 below shows a block diagram of the DPE implemented using three Queued-Stack (QS) elements and one output FIFO. The input QS elements are used to receive channel data from two sources or they can be configured such that one QS element is receiving data while the other is processing data from a previous transaction. The Result-QS is used to store the results of the datapath operations.

The DPE is implemented using a parameterized synthesized model where the width and depth of the stacks, functional units, and data-paths are determined during algorithmic development time. For systems that are composed of multiple DPEs, it is feasible for each DPE to be configured for a task or group of tasks during the synthesis process by selecting the optimal parameters.

The microcode storage is implemented using a standard single port ROM or RAM/WCS memory compiler. The RAM/WCS configuration is useful for systems where the microcode needs to be updated from an external source such as flash memory [20].

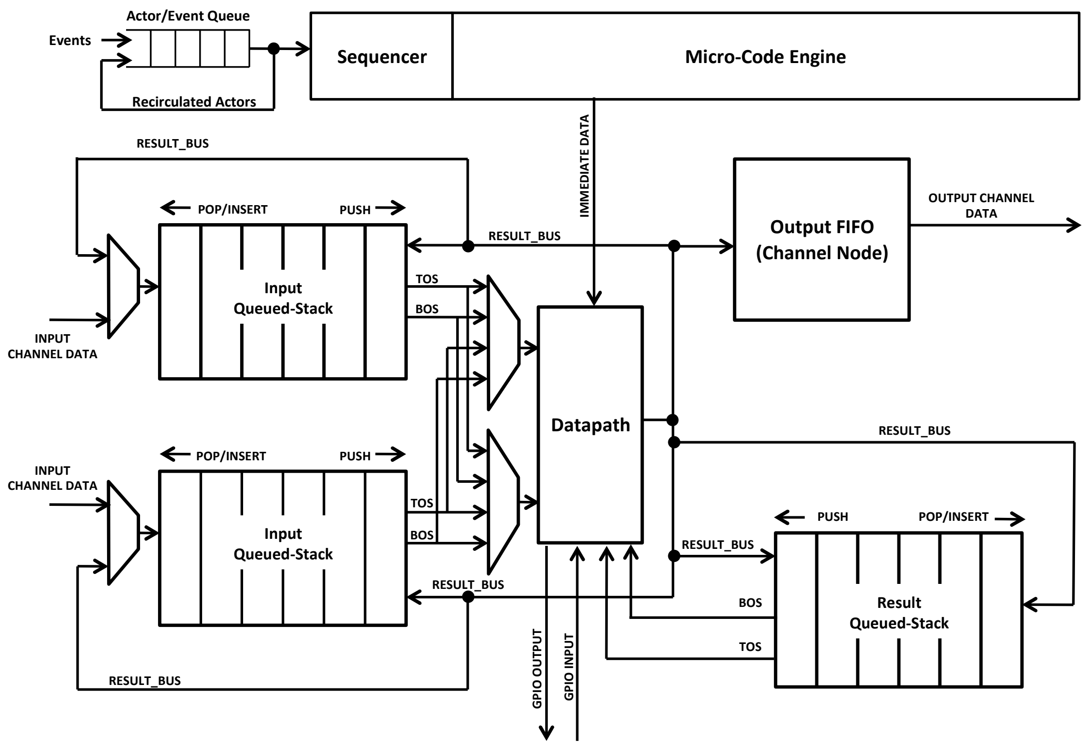


Figure 10. Dataflow Processing Element block diagram

4. SYSTEM COMPOSABILITY

Composability provides the ability to select “composable” elements and assemble them into a topology needed for a specific algorithm. For an SDF element to be composable it must be modular (self contained) and can be deployed independently. It must also be stateless which means that it treats each request (or firing) as an independent transaction, unrelated to any previous request [21]. The composition rules for this platform are:

Rule 1: All inputs to an element (actor) will have a FIFO queue. The output can have a queue to satisfy the need of Rule #3 below.

Rule 2: All data propagates through a dataflow network via channels. Note: channel nodes convert data streams as they pass through the network, e.g. Serial-Parallel, Parallel-Serial, Stream-FLIT, FLIT-Stream, etc.

Rule 3: For PUSH MODE operation, Reads to the FIFO will block, however Writes will not. For PULL MODE operation the inverse is true. This platform is designed to support both PUSH and PULL mode operation.

Rule 4: The composed system will be determinate which requires that each actor is functional and that the set of firing rules are sequential. “Functional” means that an actor firing lacks side effects and that the outputtokens are purely a function of the input tokens consumed in that firing.

Rule 5: Elements can be software routines. Rule #4 states that these routines can be moved to alternate computational engines and execute without modification.

The composable platform is implemented using an event driven synchronous dataflow architecture. The system is “composed” by instantiating dataflow-processing elements (DPE) that are connected via channels. The DPEs provide key functional services (actors) in the dataflow system. In the current implementation the inputs to the actors are modeled as bounded queues as it is possible to determine a-priori what the depth of each queue needs to be during algorithmic mapping and system level modeling.

Figure 11 below shows an example of a composed system with functional and channel nodes. In this example the channel nodes are used to convert data from the sensor element into data tokens that are forwarded to the DPEs. The Communications Element is an event driven functional node and follows the composability rules described above.

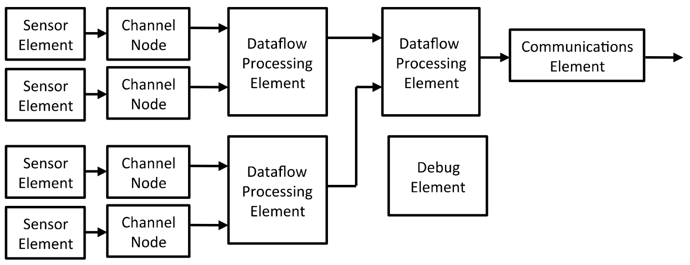


Figure 11. Composed system showing functional and channel nodes

Figure 12 below shows a sensor system topology where the channel nodes are configured as routers. The channel routing nodes route tokens through the network in a predefined pattern. The routing patterns are loaded during system initialization and are static. These types of network topologies provide flexibility in building a wide range of sensor platforms at the expense of increased energy usage [22].

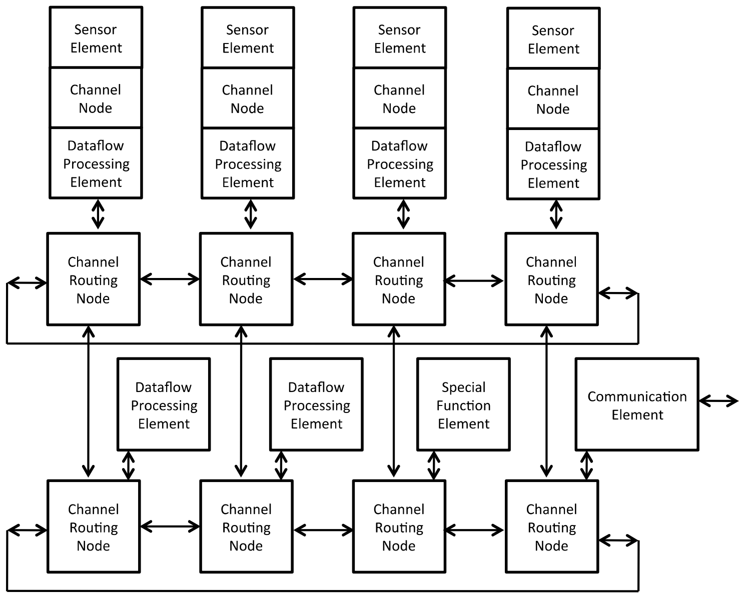


Figure 12. Composed system showing channel nodes configured as routers

5. MODELING and PROGRAMMING

The CSP is modeled, analyzed and programmed using MATLAB/Simulink and SimEvents from MathWorks [23]. SimEvents is an event-based simulator that works in conjunction with Simulink to model both time based and event driven systems. The sensor and ADC sub-system can be described in MATLAB or built from Simulink library models. The output of the ADC is converted into a signal-event that is processed by the SimEvents simulator. SimEvents does not do computational simulation but rather simulates tokens (entities) propagating through the SDF network. Each resource in the network can be instrumented to determine if there are any errors as the tokens propagate through the network. Additionally the instrumentation enables debug capability by providing visibility to various parameters in a network resource.

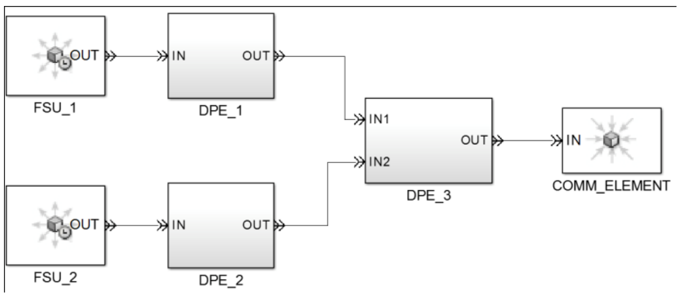


Figure 13. CSP SimEvents Model

Figure 13 above shows a CSP SimEvents model with two Functional Service Units (FSU) and three Dataflow Processing Elements. The FSUs are token generators that launch tokens into the network. Each token has two attributes as shown above in Figure 14. The DPE's process the tokens and then outputs them to a communications element that is modeled as a token sink.

DPE_1 and DPE_2 are modeled with a single queue and a single actor as shown below in Figure 14. Note the instrumentation ports on the actor. These are used to determine optimal resource allocation for the single queue DPE.

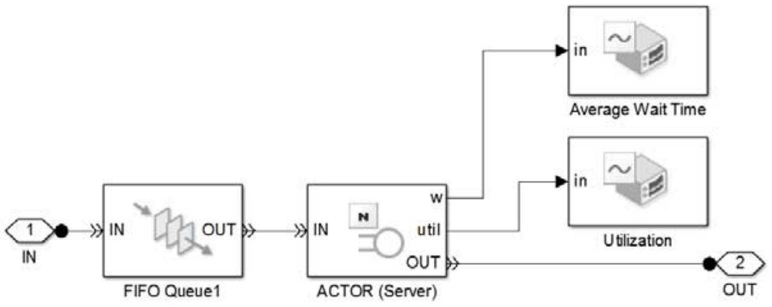


Figure 14. Single queue DPE model

DPE_3 is modeled with two queues, a token combiner, token consumer and a single actor as shown below in Figure 15.

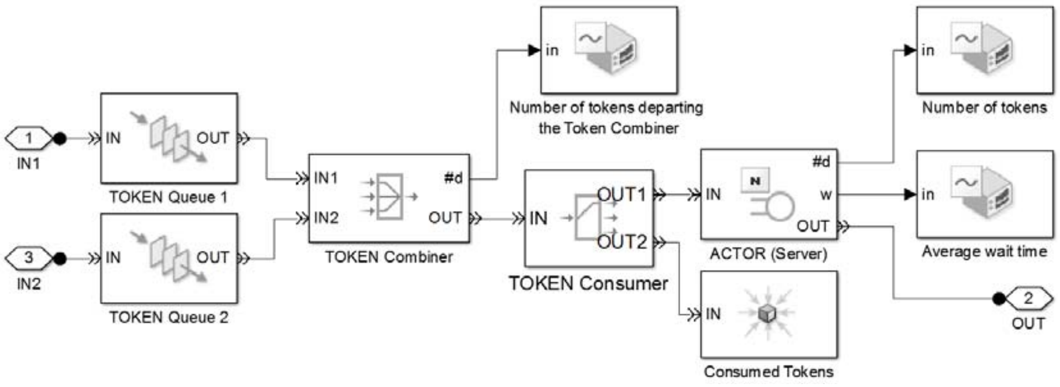


Figure 15. Multiple Queue DPE model

The CSP is programmed by selecting actors from a library and instantiating them on a DPE. A broad selection of actors are available including: digital filtering functions, decimation, linearization, averaging, successive approximation, data compression/suppression, data fusion, period measurements, threshold detection, edge detection, etc. Each actor is implemented as a standalone microcoded routine that terminates once it completes and then waits for the next event. The actors are loaded into an actor/event queue on the DPE and the relative order of how the actors are processed can be dynamically changed by reentering or reordering the actors in the queue.

6. SUMMARY

This paper presents a unique system implementation of a low energy cognitive sensor platform. The system uses a synchronous dataflow paradigm to process data tokens. The dataflow-processing element utilizes a novel queued-stack architecture that directly supports the SDF protocol. The CSP provides optimal energy-performance by tuning the micro-architecture of the computational element(s) to directly implement the SDF channel interface protocol while the SDF “actor” based programming paradigm is tuned for reactive sensor system applications. The composability capabilities of the platform provide the mechanism to build complex systems that adhere to the SDF protocol.A high level modeling environment is used to compose the SDF network and program the individual dataflow processing elements.

REFERENCES

[1] A. Howard and E. Tunstel, “*Development of Cognitive Sensors*”, NASA TECH BRIEF Vol. 26, No. 4.
[2] J. Koomey, et al, “Implications of Historical Trends in the Electrical Efficiency of Computing”, *Annals of the History of Computing, IEEE*, Volume: 33 Issue: 3, pp. 46 – 54, March 2011.
[3] D. Harel and A. Pnueli, “*On the Development of Reactive Systems. In Logics and Models of Concurrent Systems*”, (K. R. Apt, ed.), NATO ASI Series, Vol. F-13, Springer-Verlag, New York, pp. 477-498, 1985.
[4] B. Lee. 2000, “*Specifications and Design of Reactive Systems*”, Ph.D. Thesis. Memorandum UCB/ERL M00/29, Electronics Research Laboratory, University of California, Berkeley, May 2000.
[5] J. Elson, J. and D. Estrin, “*Fine-Grained Network Time Synchronization using Reference Broadcast*”, The Fifth Symposium on Operating Systems Design and Implementation (OSDI), pp. 147-163, December 2002.
[6] S. Ganeriwawal, R. Kumar, M. Srivastava, “*Timing-Sync Protocol for Sensor Networks*”, The First ACM Conference on Embedded Networked Sensor Systems (SenSys), p. 138-149, November 2003.
[7] M. Maroti, B. Kusy, G. Simon, A. Ledeczi, “*The Flooding Synchronization Protocol*”, Proc. Of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys), November 2004.
[8] C. Lee, “Fuzzy Logic Control Systems”, *IEEE Trans. on Systems, Man, and Cybernetics, SMC*, Vol. 20, No. 2, 1990, pp. 404-435.
[9] J. Hauser, J. Wawrzynek, “*GARP: a MIPS processor with a reconfigurable coprocessor*”, Field-Programmable Custom Computing Machines, 1997. Proceedings, The 5th Annual IEEE Symposium, Vol., no., pp.12-21, 16-18 Apr 1997.
[10] Q. Zhang, A. Kokkeler and G. Smit, “Dynamically reconfigurable FFTs for a Cognitive Radio on a multiprocessor platform”, *International Conference on Engineering of Reconfigurable Systems and Algorithms 2008 (ERSA2008)*, Jul. 2008.
[11] CPU-12 Reference Manual, Rev. 4.0, Chapter 9, pp. 341-380, Motorola Inc., May 2003
[12] D. Hall and J. Llinas, “*An introduction to Multi-sensor data fusion*”, Proceedings of the IEEE, vol. 85, No. 1, January 1997, pp. 6-23.
[13] R. Gibson, D. L. Hall, and J. A. Stover, “*An autonomous fuzzy logic architecture for multi-sensor data fusion*”, in Proc. 1994 IEEE Conf. on Multi-Sensor Fusion and Integration for Intelligent Systems, Las Vegas, NV, pp. 143–150, Oct. 1994
[14] M. Akhoudi, E. Valav, “*Multi-Sensor Fuzzy Data Fusion Using Sensors with Different Characteristics*”, Unpublished, submitted to The CSI Journal on Computer Science and Engineering (JCSE)
[15] Y. Vershinin, “*A Data Fusion Algorithm for Multi-sensor Systems*” Proc. of ISIF, Jul. 2002, pp. 341-345.
[16] E. Punskaya, “*Bayesian approaches to multi-sensor data fusion*” Master’s thesis, Cambridge University Engineering Department, 1999.
[17] P. Escamilla-Ambrosio and N. Mort, “*Hybrid Kalman Filter-Fuzzy Logic Adaptive Multi-sensor Data Fusion Architecture*”, Proc. of The IEEE Conference on Decision and Control, pp. 5215-5220, 2003
[18] A. Manickam, A. Chevalier, M. McDermott, A. D. Ellington, A. Hassibi, “*A CMOS electrochemical impedance spectroscopy biosensor array for label-free biomolecular detection*”, ISSCC 2010, pp. 130-131
[19] M. McDermott, “QueuedStack Dataflow Processing Element for a Cognitive Sensor Platform”, *International Journal of Reconfigurable and Embedded Systems (IJRES)*, Vol 1, No 3, pp. 75-86, 2012
[20] P. Koopman, “*Writable instruction set, stack oriented computers*”, Proceedings of the 1987 Rochester Forth Conference.
[21] B. Lee, and A. R. Hurson, “*Issues in dataflow computing*”, In Adv. in Computing, Vol. 37, pp. 285-333, 1993
[22] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey, A. Sangiovanni-Vincentelli, *Addressing the System-on-a-Chip Interconnect Woes Through Communication-Based Design*. In DAC 2001, June 18-22, 2001
[23] MATLAB: SimEvents®: www.mathworks.com/products/simevents/

BIOGRAPHY OF AUTHOR

Mark McDermott received his MSE and Ph.D. from the University of Texas at Austin where where he teaches graduate level courses in VLSI design, Embedded System design and System-on-Chip design. He is a registered professional engineer and a member of the IEEE, ACM and TSPE. His research interests are in the areas low energy embedded system design and VLSI design. He has 19 patents and 10 publications in the areas of VLSI system design and engineering education.