Dataflow Processing: Modeling to Implementation

Mark McDermott

Fall 2019

Agenda

- Principles of Dataflow
- Mapping to MATLAB/SimEvents
- Dataflow Processor Example

Dataflow*

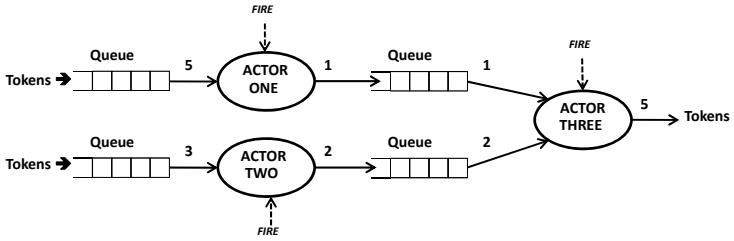
- **Definitions**
  - A token is a data value or data structure
  - A signal is a sequence of tokens
  - A node maps (converts) input tokens onto output tokens
  - Set of firing rules specify when a node can fire
  - A firing of a node consumes input tokens and produces output tokens
  - A sequence of firings is a dataflow process

* Dataflow is loosely based on the premise that changing the value of a variable should automatically force recalculation of the values of other variables much like a spreadsheet.

Dataflow Networks

- A dataflow network is a collection of functional (stateless) nodes which are connected and communicate over unbounded FIFO queues
- Nodes are commonly called actors
  - actors perform computation
- The bits of information that are communicated over the queues are commonly called tokens
- Unbounded FIFO perform communication via sequences of tokens carrying values
  - integer, float, fixed-point
  - matrix of integer, float, fixed-point
  - image of pixels
- Determinacy
  - unique output sequences given unique input sequences
  - sufficient condition: blocking read (actor cannot test input queues for emptiness)

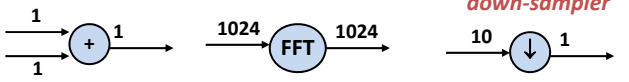
Example of a Data Flow Network



Examples of Data Flow Actors

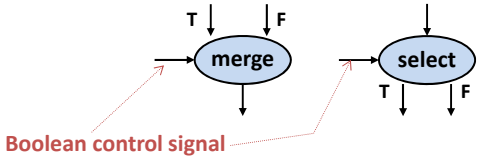
SDF: Synchronous (Static) Data Flow

Fixed input and output tokens



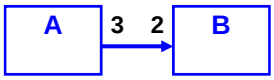
BDF: Boolean Data Flow

Control token determines consumed and produced tokens



Synchronous Dataflow

- Untimed
- Arcs: one-way first-in first-out (FIFO) queues
- Nodes/Actors: functional blocks
  - Source nodes always enabled
  - Others enabled when enough samples/tokens are on all inputs

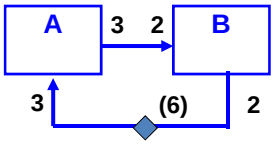


- Node execution
  - Consumes same fixed number of samples/tokens on each input arc
  - Produces same fixed number of tokens on each output arc
  - Consumed data is de-queued from arc
- Flow of data through graph does not depend on data values

Synchronous Dataflow (cont.)

- Delay of (*n*) samples
  - *n* samples initially in FIFO queue
- Systems are *determinate*
  - Execution in sequence or parallel has same outcome (predictable)
- Systems can be *statically analyzed*
  - Check for "sampling rate" consistency
  - Determine/optimize FIFO queue sizes at compile time
- Models systems with rational rate changes

Periodic schedule fires A twice & B thrice, e.g. AABBB or ABABB



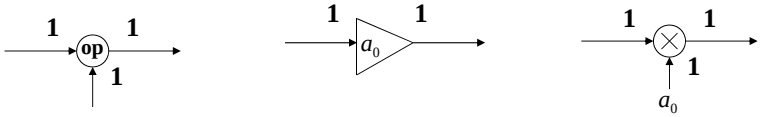
Nodes are not multirate but graph is!

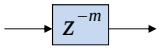
Dataflow Models in Design Tools

Design Tool	Dataflow Model(s)	Example Applications
Matlab/SimEvents	Synchronous and Timed Synchronous Dataflow	Mixed analog, digital, and RF communication systems
National Instruments LabVIEW	Homogeneous Dynamic Dataflow (G)	Periodic and aperiodic digital systems
Synopsys CoCentric System Design Studio	Cyclo-static Dataflow	Periodic digital systems, e.g. transceivers & mp3 decoders
UC Berkeley Ptolemy	Synchronous and Dynamic Dataflow	Periodic and aperiodic digital systems

Homogeneous Operations

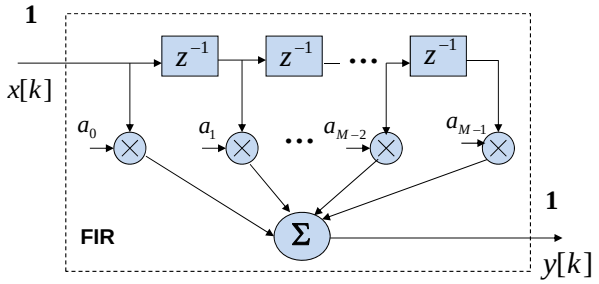
- **Pointwise arithmetic operations (addition, etc.)**



- **Delay by *m* samples**  ***property of SDF arc***

Finite impulse response (FIR) filter:

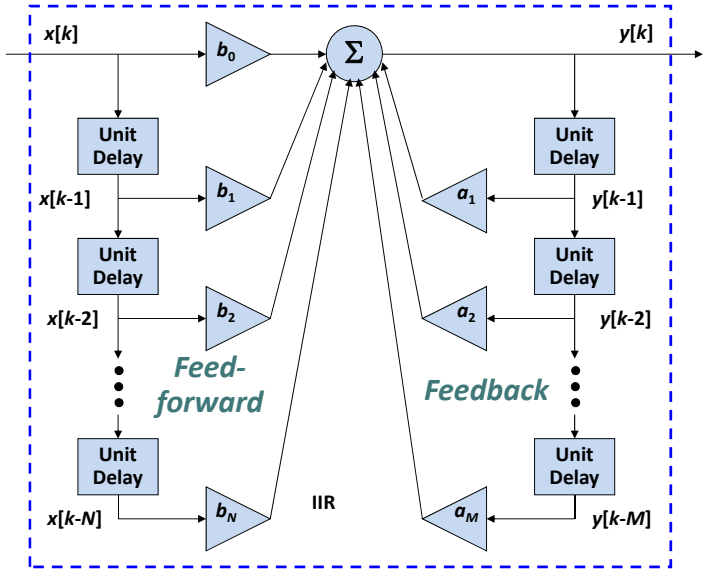
$$y[k] = \sum_{m=0}^{M-1} a_m \, x[k-m]$$



Homogeneous Operations

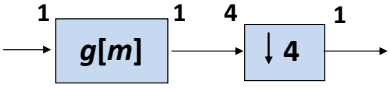
Infinite impulse response (IIR) filter:

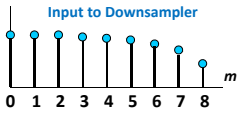
$$y[k] = \sum_{n=0}^{N} b_n\, x[k-n] + \sum_{m=1}^{M} a_m\, y[k-m]$$

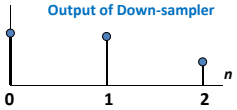


Decreasing Sampling Rate

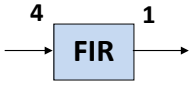
- **Finite impulse response (FIR) filter *g*[*m*]**
  - Typically a lowpass filter
  - Enforces sampling theorem
- **Down-sampling by *L* denoted as $\downarrow L$**
  - Inputs *L* samples
  - Outputs first sample and discards *L*-1 samples
  - Decreases sampling rate by factor of *L*





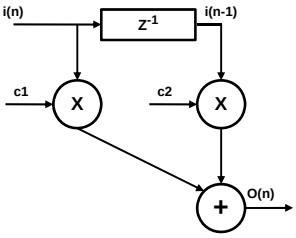


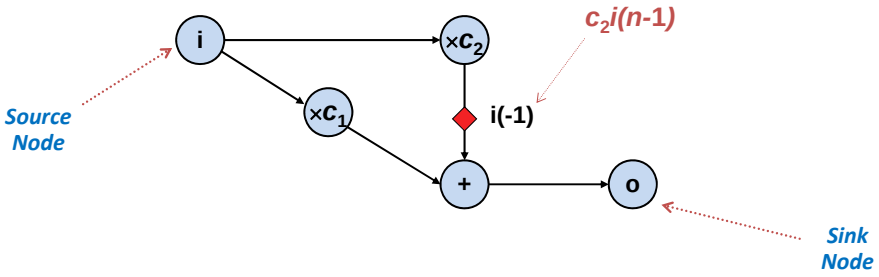
- **Sometimes combined into rate changing FIR block**



Example: 2-Stage FIR Filter

- **FIR Filter**
  - single input sequence *i(n)*
  - single output sequence *o(n)*
  - $o(n) = c_1 i(n) + c_2 i(n\text{-}1)$

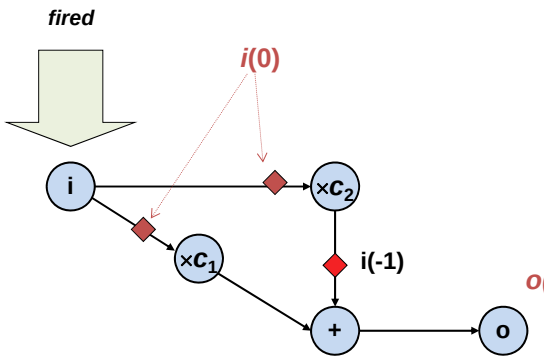




(each actor consumes one token and produces one token)

Example: 2-Stage FIR Filter

- **A sequence of firings is a dataflow process**

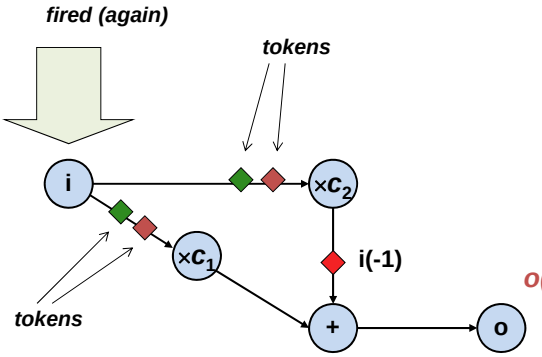


$o(n) = c_1 i(n) + c_2 i(n\text{-}1)$

(each actor consumes one token and produces one token)

Example: 2-Stage FIR Filter

- A sequence of firings is a dataflow process

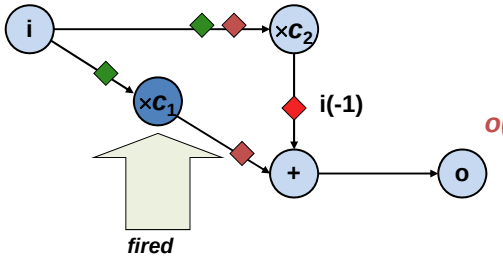


$o(n) = c_1 i(n) + c_2 i(n\text{-}1)$

(each actor consumes one token and produces one token)

Example: 2-Stage FIR Filter

- A sequence of firings is a dataflow process

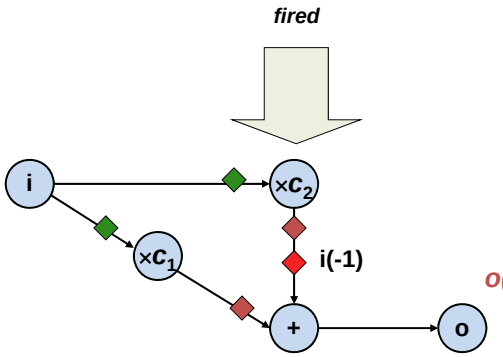


$o(n) = c_1 i(n) + c_2 i(n\text{-}1)$

(each actor consumes one token and produces one token)

Example: 2-Stage FIR Filter

- A sequence of firings is a dataflow process

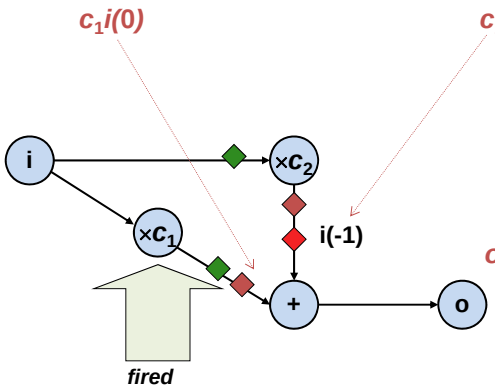


$o(n) = c_1 i(n) + c_2 i(n\text{-}1)$

(each actor consumes one token and produces one token)

Example: 2-Stage FIR Filter

- A sequence of firings is a dataflow process

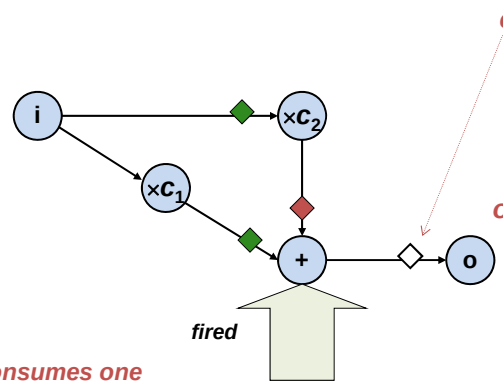


$o(n) = c_1 i(n) + c_2 i(n\text{-}1)$

(each actor consumes one token and produces one token)

Example: 2-Stage FIR Filter

- A sequence of firings is a dataflow process

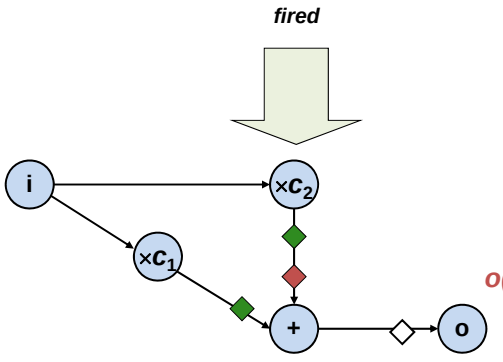


$o(n) = c_1 i(n) + c_2 i(n\text{-}1)$

(each actor consumes one token and produces one token)

Example: 2-Stage FIR Filter

- A sequence of firings is a dataflow process

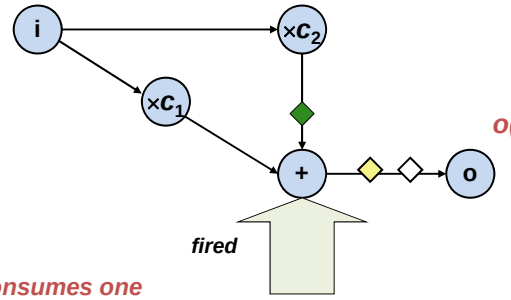


$o(n) = c_1 i(n) + c_2 i(n\text{-}1)$

(each actor consumes one token and produces one token)

Example: 2-Stage FIR Filter

- A sequence of firings is a dataflow process

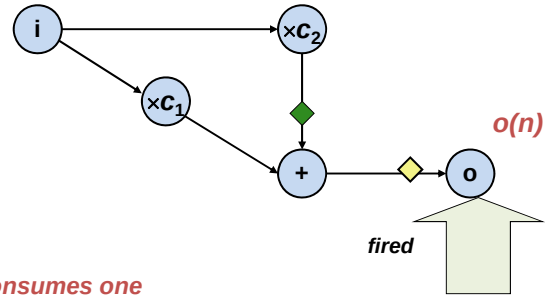


$o(n) = c_1 i(n) + c_2 i(n\text{-}1)$

(each actor consumes one token and produces one token)

Example: 2-Stage FIR Filter

- A sequence of firings is a dataflow process

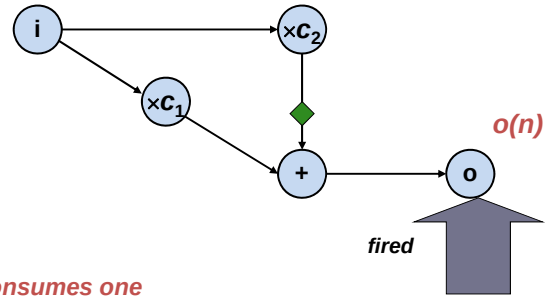


$o(n) = c_1 i(n) + c_2 i(n\text{-}1)$

(each actor consumes one token and produces one token)

Example: 2-Stage FIR Filter

- A sequence of firings is a dataflow process



$o(n) = c_1 i(n) + c_2 i(n\text{-}1)$

(each actor consumes one token and produces one token)

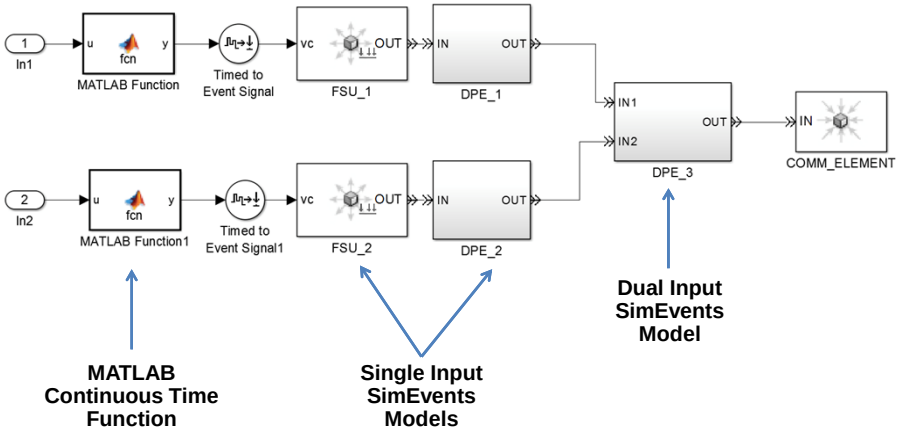
Agenda

- Principles of Dataflow
- Mapping to MATLAB/SimEvents
- Dataflow Processor Example
  - Cognitive Sensor Platform

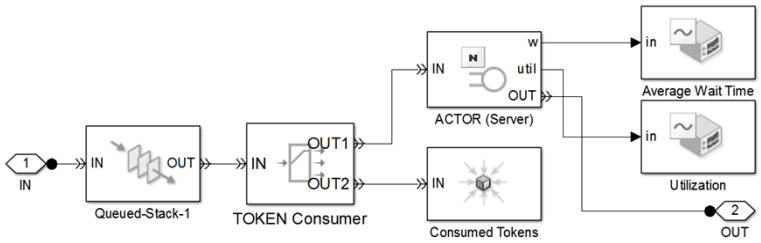
High Level Modeling in SimEvents

- **SimEvents is an event-based simulator from Math Works.**
  - Used in conjunction with Matlab/Simulink to model both continuous time based systems and event-driven systems.
- **Sensor and ADC sub-system are described in Matlab or built from Simulink library models.**
  - Output of the ADC is converted into a signal-event that is processed by the SimEvents simulator.
  - SimEvents does not perform a computational simulation but rather simulates entities (tokens) propagating through the SDF network.
- **Each resource in the network can be instrumented to determine if there are any errors as the tokens propagate.**

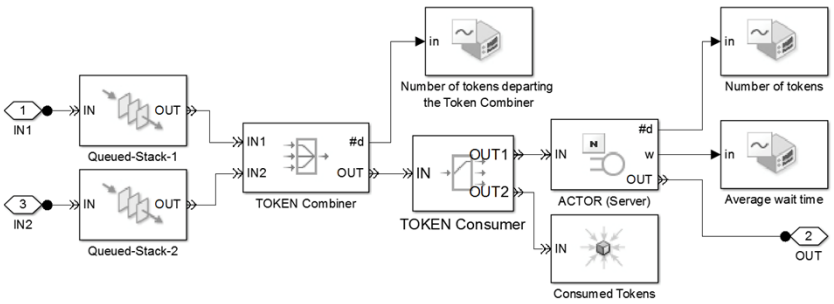
Multi-DPE Matlab/SimEvents Model



SimEvents Model for a Single Input DPE



SimEvents Model for a multiple input DPE



Agenda

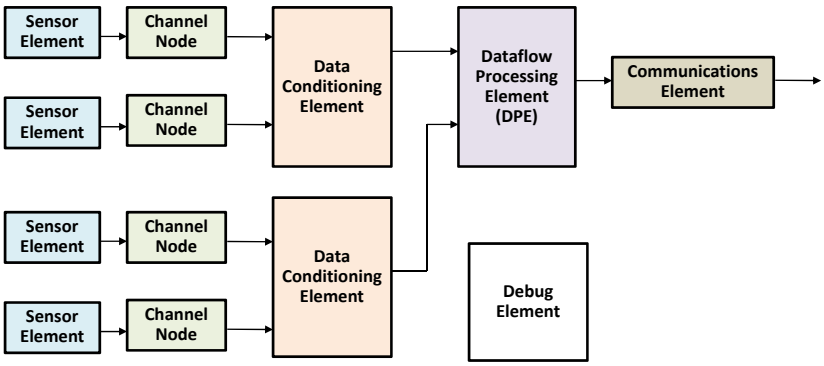
- Principles of Dataflow
- Mapping to MATLAB/SimEvents
- **Dataflow Processor Example**
  - **Cognitive Sensor Platform**

Typical Actors

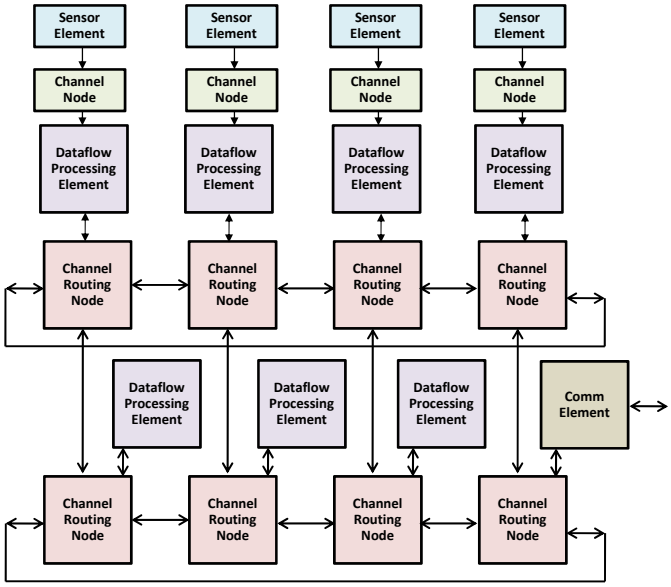
Actor	# Input Tokens	# Output Tokens	# DPE Input Channels
IIR & FIR Filter	1	1	1
Decimation Filter	2:n	1	1:n
Average	2:n	1:n	1:n
Data Compression	2:n	1:n	1:n
Data Fusion	2:n	1:n	1:n
Period Measurement	2	1	1:n
Threshold Detection	1	1	1
Event Triggering	1:n	1:n	1:n

Basic SDF System

- The channel interface protocol provides the capability to compose large systems from smaller building blocks.
- The following example shows a four sensor node system.



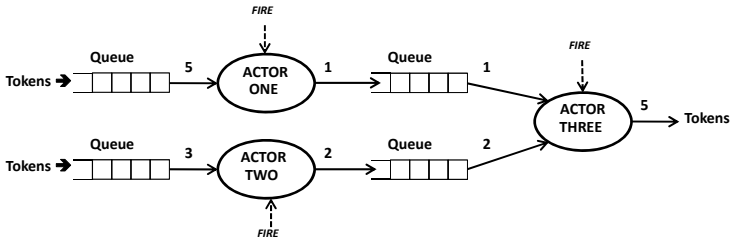
SDF System Configured for NOC operation



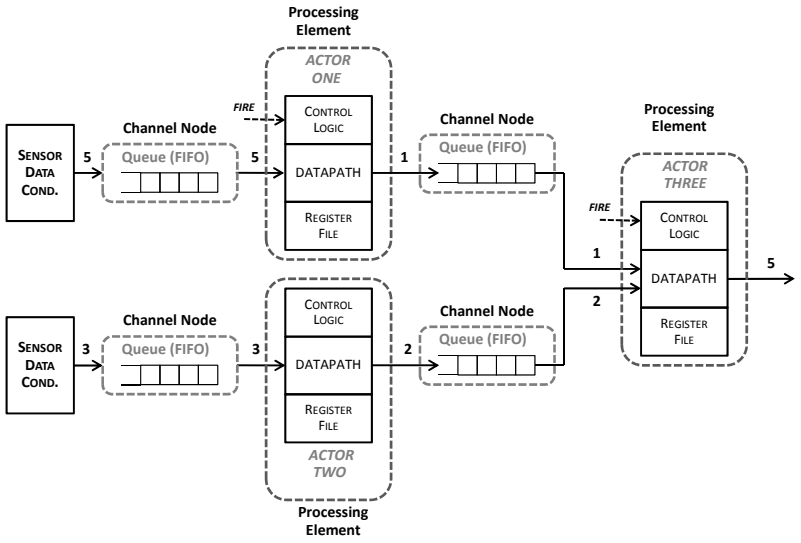
SDF Composability

- For a component to be composable it must be modular (self-contained) and can be deployed independently.
- Components are connected via channels.
- Inputs to components have FIFOs that buffer tokens.
- Must be stateless which means that it treats each request (or firing) as an independent transaction, unrelated to any previous request.
- Components can be software or hardware as long composability rules are adhered to.
- Components may contain many components as long as all of the composability rules are adhered to.

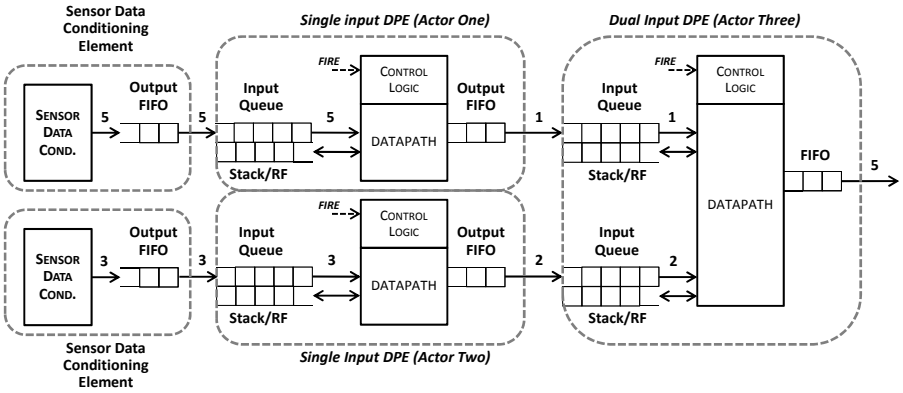
SDF Composability – Step 1



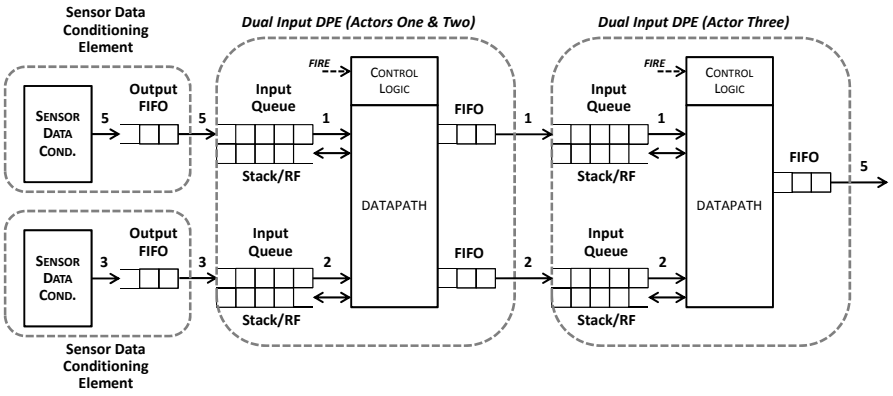
SDF Composability – Step 2



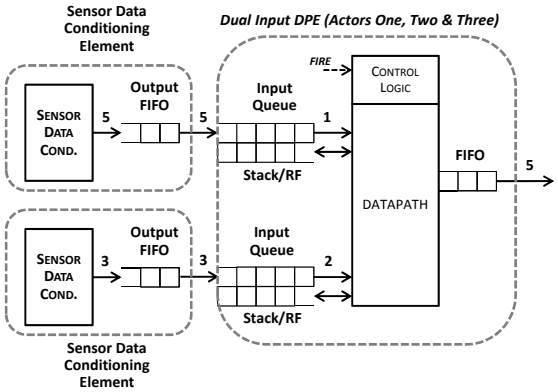
SDF Composability – Step 3



SDF Composability – Step 4

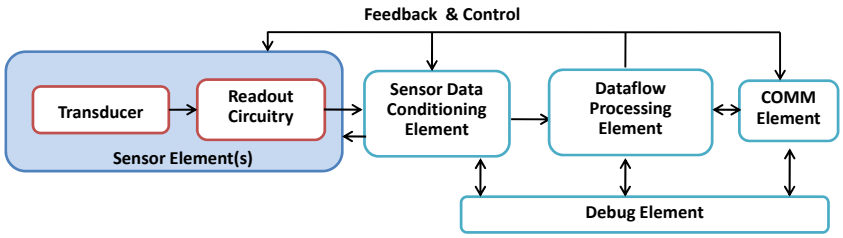


SDF Composability – Step 5



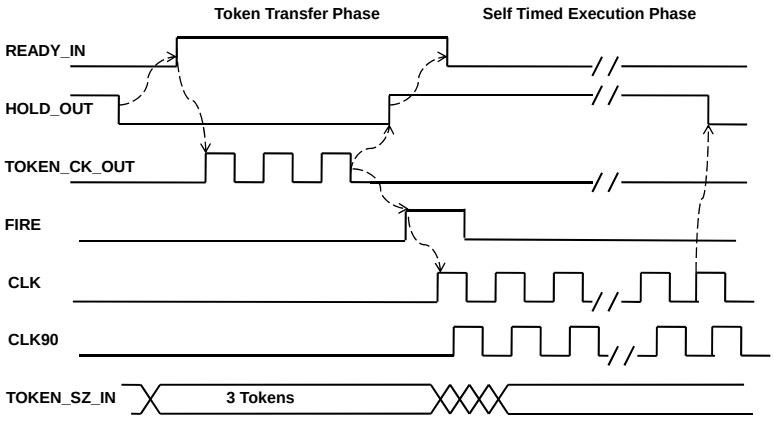
Cognitive Sensor Platform (CSP) Architecture

- **The CSP is composed five basic functional elements:**
  - Transducer and readout circuitry
  - Sensor data conditioning, time stamping, etc.
  - SDF processing element
  - Communication element
  - Optional debug and control element



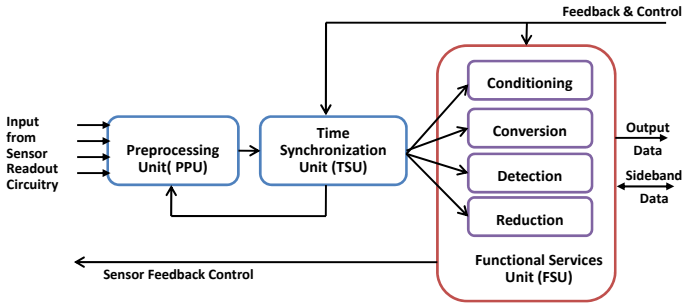
Self timed execution

- **The sequencing of the channel or service node is controlled via an internal self-timed clocking mechanism.**



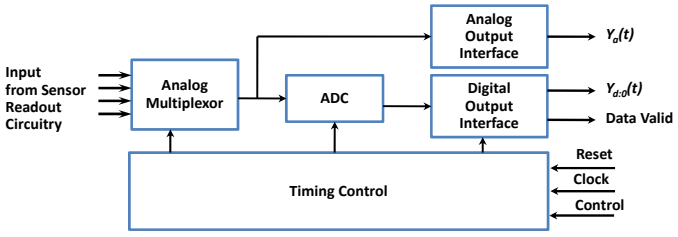
SDC Element

- **The SDC contains three basic units:**
  - Functional services unit (FSU) - performs data-flow computational tasks.
    - Can be implemented as a FSM or use a DPE
  - Preprocessing unit (PPU) - includes filters, A/D converters, etc.
  - Time synchronization unit (TSU) - performs the time stamping function and limited data conditioning functions.



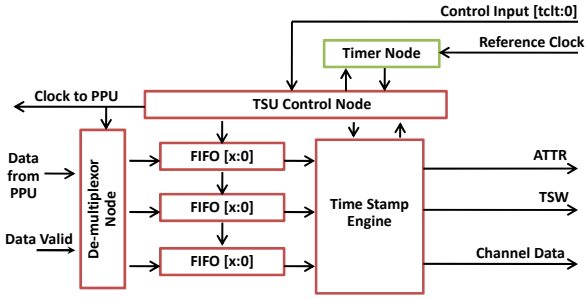
Preprocessing Unit

- **Provides typical preprocessing functions:**
  - A/D conversion
  - Filtering
- **Receives control signals from Time Synchronization Unit**
  - Adaptive control based on data attributes



Time Synchronization Unit

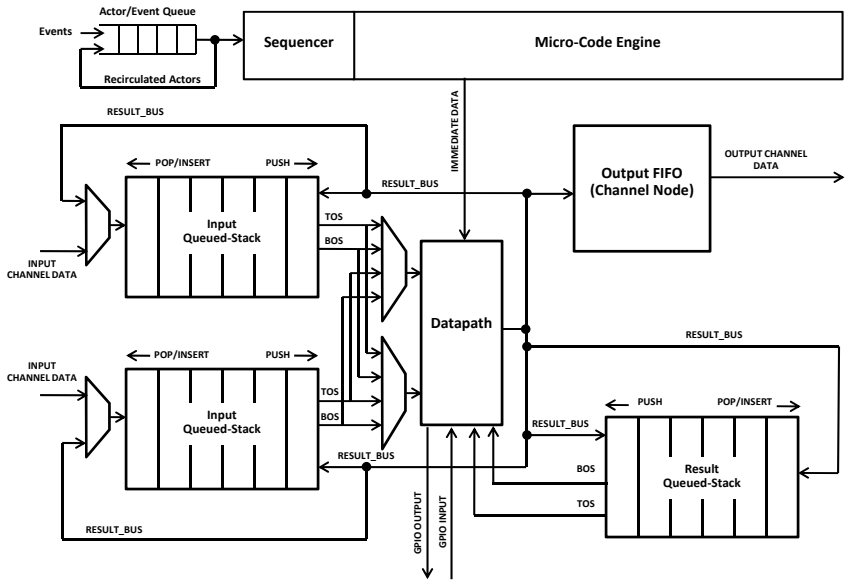
- There are five components in the TSU:
  - Demultiplexor Node - used to demultiplex the data from PPU
  - FIFO Queues – stores data and time stamp information
  - Timer Node – generates the timing reference for control unit
  - Time Stamp Engine – converts queued data for use by the FSU
  - Control Node – FSM based controller



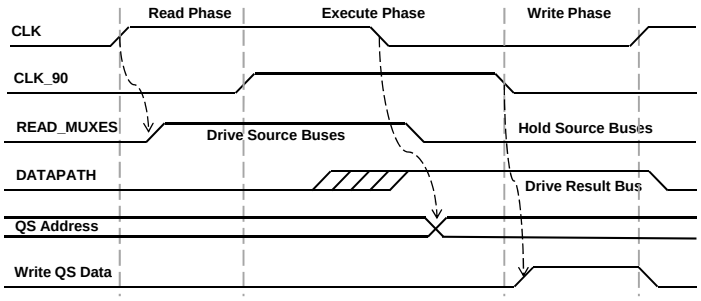
Dataflow Processing Element

- Stack-based microcoded engine
  - Repeat execution
  - Nested looping
  - Conditional execution (non-deterministic operation)
  - Two types of conditional branching
- Single cycle fetch-scale-multiply-accumulate-writeback
- Actor/event queue
- Fuzzy-logic acceleration
- Reprogrammable content-addressable lookup table
  - Used for modified antecedent-consequence rules in the fuzzy logic machine.
- Variable data precision

Dataflow Processing Element (DPE)

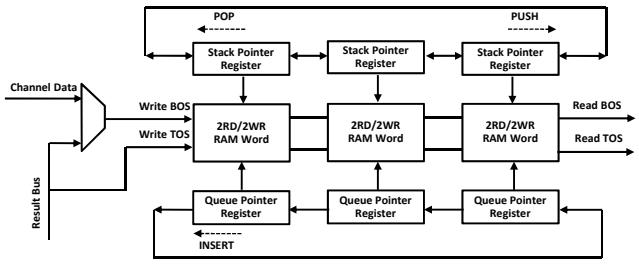


DPE Timing



Single cycle Read-Execute-Write operation using phase shifted clocks to delineate sub-cycle operations.

Queued-Stack (QS) element

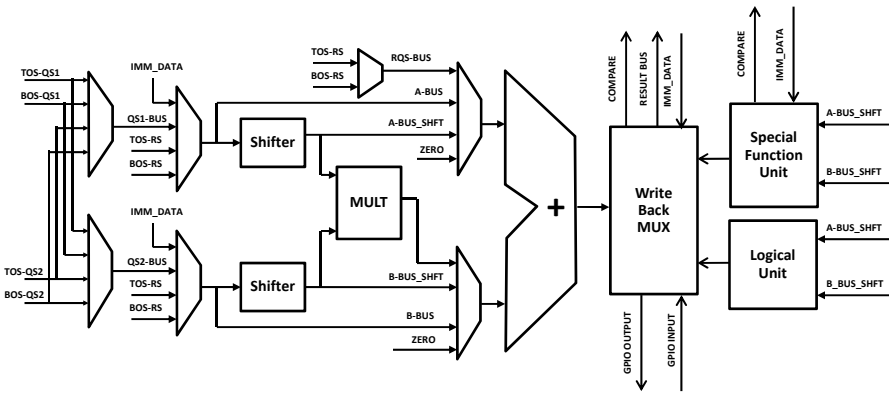


- Merges channel FIFO with datapath stack based register file (LIFO).
- Separate one-hot FIFO/LIFO pointers implemented with shift registers.
  - Minimizes decoding & glitching power
- All channel inputs are inserted into the bottom of stack (BOS). Datapath operations can push data on the top of stack (TOS) and/or insert to the BOS.
- FIFO controller "fires" DPE when requisite channel data is inserted.

QS Operations

Operation	Description
PUSH	Rotate TOS pointer right and write result-bus value to new TOS
POP	Rotate TOS pointer left (w/o write)
POP_WR	Rotate TOS pointer left and write result-bus value to new TOS
INS	Rotate BOS pointer left and write result-bus value to BOS
INS_NW	Rotate BOS pointer left (w/o write)
PUSH_NW	Rotate TOS pointer right (w/o write)
TOP	Write result bus value to TOS w/o rotating pointer
BOT	Write result bus value to BOS w/o rotating pointer
TOP_BOT	Write result bus value to TOS/BOS w/o rotating pointers
PUSH_INS	Rotate both pointers and write result-bus value to TOS/BOS
POP_BOT	Rotate TOS pointer left and write result-bus to BOS
POP_INS	Rotate TOS pointer left, rotate BOS left and write result-bus to new BOS
POP_WR_BOT	Rotate TOS pointer left and write result-bus to BOS and to new TOS
PUSH_NW_BOT	Rotate TOS pointer right and write result-bus to BOS
TOP_INS	Rotate BOS pointer left and write result-bus to TOS and to new BOS
NOP	No Operation

Datapath



Shifter Input	Multiplier/Log/SFU Input	Adder Input
TOS/BOS IQS1	SHIFTER_A/SHIFTER_B	SHIFTER_A/SHIFTER_B
TOS/BOS IQS2	TOS/BOS IQS1	TOS/BOS IQS1
TOS/BOS RQS	TOS/BOS IQS2	TOS/BOS IQS2
IMMEDIATE DATA	TOS/BOS RQS	TOS/BOS RQS
	IMMEDIATE DATA	IMMEDIATE DATA
		MULTIPLIER

Datapath Operations

SHIFT MODE A <44>	SHIFT A <38:34>	SHIFT TC A <32>	DATA TC <19>	MSB OP	OPERATION
1	0-31	0	-	-	LEFT ARITHMETIC SHIFT, LOGIC 0 PADDING
1	0-31	1	-	0	LEFT ARITHMETIC SHIFT, LOGIC 0 PADDING
1	0-31	1	0	0	RIGHT ARITHMETIC SHIFT, LOGIC 0 PADDING
1	0-31	1	1	1	RIGHT ARITHMETIC SHIFT, SIGN EXT. PADDING
0	0-31	0	-	0	LEFT BARREL SHIFT
0	0-31	1	-	1	RIGHT BARREL SHIFT

SHIFT MODE B <45>	SHIFT B <39:43>	SHIFT TC B <33>	DATA TC <19>	MSB OP	OPERATION
1	0-31	0	-	-	LEFT ARITHMETIC SHIFT, LOGIC 0 PADDING
1	0-31	1	-	0	LEFT ARITHMETIC SHIFT, LOGIC 0 PADDING
1	0-31	1	0	0	RIGHT ARITHMETIC SHIFT, LOGIC 0 PADDING
1	0-31	1	1	1	RIGHT ARITHMETIC SHIFT, SIGN EXT. PADDING
0	0-31	0	-	0	LEFT BARREL SHIFT
0	0-31	1	-	1	RIGHT BARREL SHIFT

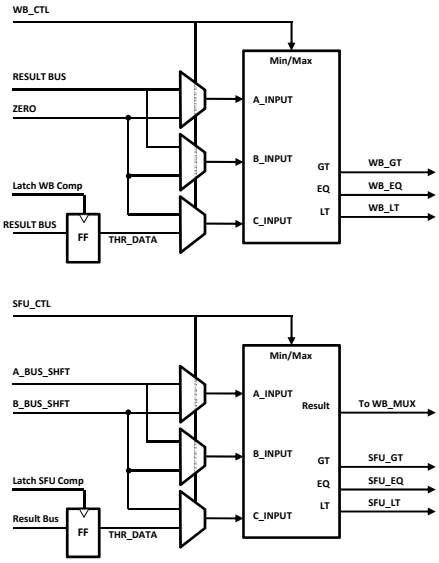
MULT MODE <31>	DATA TC <19>	OPERATION
0	-	NO OPERATION
1	0	UNSIGNED MULTIPLY
1	1	SIGNED MULTIPLY

ADD/SUB <28>	SAT <30>	DATA TC <19>	OPERATION
0	0	0	UNSIGNED ADD
0	0	1	SIGNED ADD
0	1	0	SATURATED UNSIGNED ADD
0	1	1	SATURATED SIGNED ADD
1	0	0	UNSIGNED SUBTRACT
1	0	1	SIGNED SUBTRACT
1	1	0	SATURATED UNSIGNED SUBTRACT
1	1	1	SATURATED SIGNED SUBTRACT

Condition Code generation

- **Two compare mechanisms**
  - SMAC operations
  - Special Function Unit operations
    - NOTE: SFU used for Fuzzy Logic acceleration.

SFU_CTL	OPERATION
MIN_A_B	MIN(SHFT_A_BUS, SHFT_B_BUS, SHFT_B_BUS)
MIN_A_REF	MIN(SHFT_A_BUS, REF_DATA, REF_DATA)
MIN_B_REF	MIN(SHFT_B_BUS, REF_DATA, REF_DATA)
MIN_A_B_REF	MIN(SHFT_A_BUS, SHFT_B_BUS, REF_DATA)
MAX_A_B	MAX(SHFT_A_BUS, SHFT_B_BUS, SHFT_B_BUS)
MAX_A_REF	MAX(SHFT_A_BUS, REF_DATA, REF_DATA)
MAX_B_REF	MAX(SHFT_B_BUS, REF_DATA, REF_DATA)
MAX_A_B_REF	MAX(SHFT_A_BUS, SHFT_B_BUS, REF_DATA)



Branching & Conditional Execution

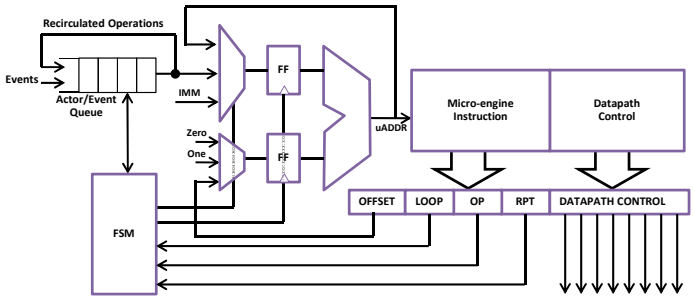
- **Nine micro-ops can be conditionally executed.**
- **Non-branching micro-ops can be repeated and/or in a nested loop.**

Micro-OP	RPT	MICRO-OPERATION
EXEC_WB_EQ	Y	EXECUTE IF WRITEBACK == THRESHOLD DATA
EXEC_WB_GT	Y	EXECUTE IF WRITEBACK > THRESHOLD DATA
EXEC_WB_LT	Y	EXECUTE IF WRITEBACK < THRESHOLD DATA
BR_WB_EQ	N	BRANCH IF WRITEBACK == THRESHOLD DATA
BR_WB_GT	N	BRANCH IF WRITEBACK > THRESHOLD DATA
BR_WB_LT	N	BRANCH IF WRITEBACK < THRESHOLD DATA
BR_SFU_GT	N	BRANCH IF SFU RESULT > THRESHOLD DATA
BR_SFU_LT	N	BRANCH IF SFU RESULT < THRESHOLD DATA
BR_SFU_EQ	N	BRANCH IF SFU RESULT == THRESHOLD DATA

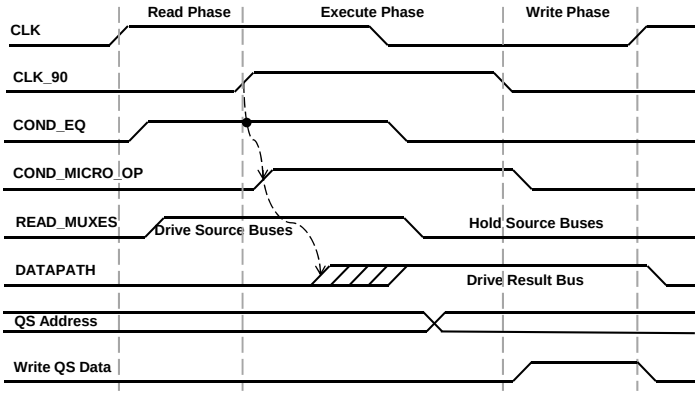
Micro-engine operations

- Un-pipelined single cycle operation.
- 1-hot control signals for maximum parallelism in each cycle.
  - Can scale, multiply and accumulate, store to 8 memory locations in one cycle.
- Supports 3-deep nested loops for matrix operations.
  - Each loop has individual repeat counts
- Most instructions can be repeated up to 2-8 times using the repeat function.
- Branch instructions use 2's complement addressing.
  - Datapath conditional branching: Branch-Equal, Branch-GT, Branch-LT
  - SFU conditional branching: Branch-Equal, Branch-GT, Branch-LT
- Jump instructions use direct addressing.

Micro-engine



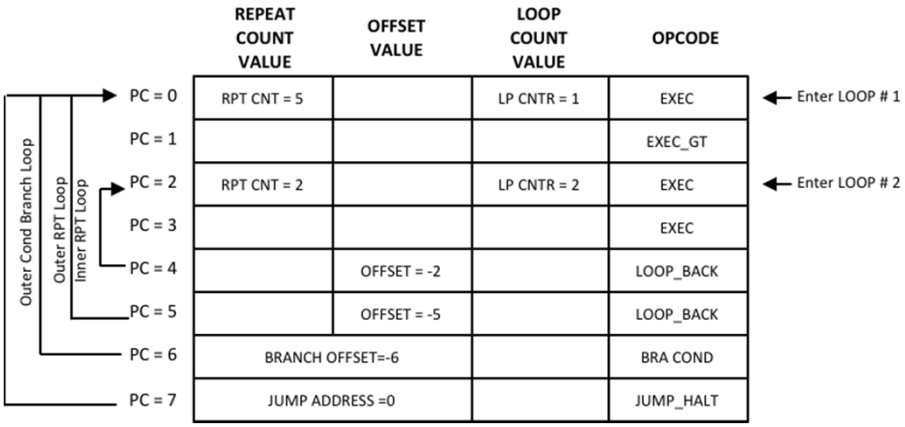
Micro-engine timing diagram



Micro-Operations

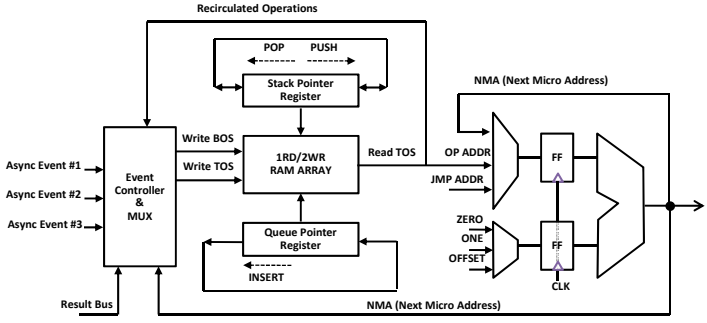
Micro-OP	RPT	MICRO-OPERATION
EXEC	Y	Unconditional Execution
EXEC_WB_EQ	Y	Execute if WRITEBACK == THRESHOLD DATA
EXEC_WB_GT	Y	Execute if WRITEBACK > THRESHOLD DATA
EXEC_WB_LT	Y	Execute if WRITEBACK < THRESHOLD DATA
WFE	N	Halt at PC+1; Wait-for-EVENT signal
JMP	N	Jump to address specified in IMMED_DATA field
JMP_HLT	N	Jump to address specified in IMMED_DATA field THEN Halt and wait for an EVENT signal
LOOP_BACK	Y	Loop back for loop # (negative offset only)
BRA	N	Branch Unconditionally
BR_WB_EQ	N	Branch if WRITEBACK == THRESHOLD DATA
BR_WB_GT	N	Branch if WRITEBACK > THRESHOLD DATA
BR_WB_LT	N	Branch if WRITEBACK < THRESHOLD DATA
BR_SFU_GT	N	Branch if SFU Result > THRESHOLD DATA
BR_SFU_LT	N	Branch if SFU Result < THRESHOLD DATA
BR_SFU_EQ	N	Branch if SFU Result == THRESHOLD DATA
TXFR	N	Transfer Output FIFO data to Channel

Nested looping example



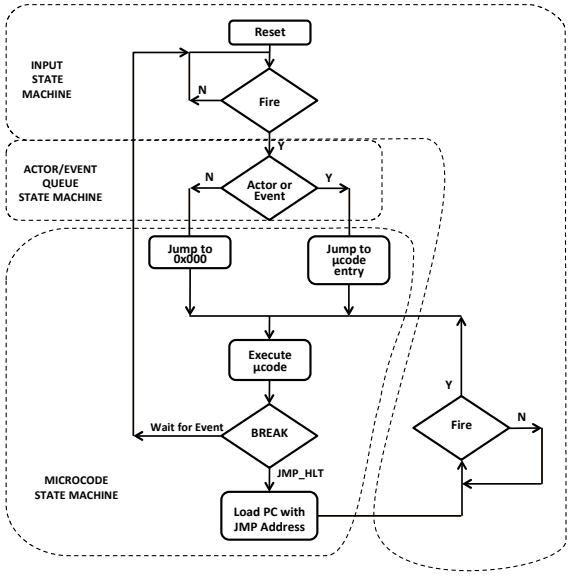
Actor/Event Queue

- **Similar to Queued-Stack.**
  - Used to store pointers to Actors (µcode routines)
  - Actors are fetched from TOS and recirculated to the BOS.
  - Contents can be dynamically changed during operation
- **Asynchronous events generate a pointer to an Actor.**
  - Pushed on TOS or inserted to BOS depending on priority.

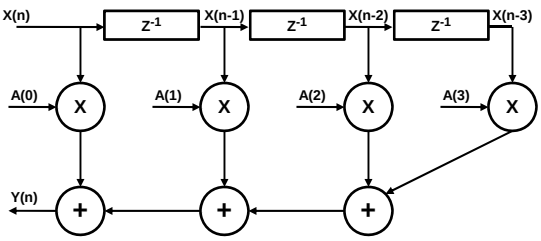


Actor/Event operational flow chart

- There are two event micro-ops:
  - JMP_HALT
  - WFE
- Both micro-ops wait for a fire signal to start actor execution



FIR Filter example




```
1:
| ZERO ZERO ADD                  // ZERO -> ACC
| INS_RQS |      :1              // INSERT AT BOS RQS

2:
| TOS_QS1 TOS_QS2 MULT           // A(3,2,1) * X(N-3,2,1)
| BOS_RQS MULT_BUS ADD           // + ACC
| BOT_RQS                        // -> ACC
| POP_QS1 | POP_QS2             // POINT AT NEW VARIABLE
| RPT=3 |          :2            // REPEAT 3 TIMES

3:
| TOS_QS1 TOS_QS2 MULT           // A(0) * X(n)
| BOS_RQS MULT_BUS ADD           // + ACC
| FIFO                          // OUTPUT Y(n) TO FIFO ELEMENT
| POP_QS1 |          :3         // CONSUME X(n) TOKEN

4:
| PUSH_QS1 | PUSH_QS2            // RESET VARIABLE POINTERS
| RPT=4 | :            4        // REPEAT 4 TIMES

5:
| JMP_HALT=1 | :5                // JUMP and HALT
```