Simulink Model Generation from an Enumeration Hybrid Automaton

Technical Report

ut-cs-12-694

Thomas Swain

5/29/2012

This work was sponsored by the UT/ORNL Joint Institute for Computational Science

Contents

List of Figures

List of Tables

Simulink™ Model Generation from an Enumeration Hybrid Automaton

Introduction

In reference [1], Hybrid Sequence-Based Specification (HSBS) is used to derive a rigorous software specification for a data acquisition processor. The purpose of [1] is to exemplify HSBS application to an existing system. The resulting specification is expressed as an enumeration hybrid automaton (EHA). To exploit fully the software reliability and development productivity benefits of HSBS, implementation of the EHA should be automated to the maximum extent possible. Fully automated implementation requires a preexisting framework that provides the following:

- I/O programming interfaces capable of accepting or generationg any of the signal types specified in the EHA.
- A set of primitive relational operators sufficient to realize all predicates associated with EHA actions.
- A set of primitive computational functions sufficient to compute all trajectories defined in the EHA.

MatLab Simulink™ is a model-based development envirionment that appears to meet these criteria for a wide range of applications. The remainder of this document describes the process of generating a Simulink™ model from the EHA in [1]. The discussion is organized in two parts:

- Extracting and grouping relevant data structures from the EHA and
- Implementing EHA data elements as a Simulink™ model.

EHA Data Structures

Signal Declarations

The following tables are based on the tables in the "Declarations" section of [1]. Modifications to the corresponding tables in [1] include (1) addition of type and range specifications for continuous signals and (2) minor redefinition of some signals resulting from a more rigorous interpretation of [2].

Table 1: Continuous Input (*U*)

Name	Variable	Units	Type	Range	Description
raw weight	rw(t)	A/D counts	integer	[0, 4095]	Weight signal from load cell

Table 2: Autonomous Signals (*X*)

Name	Variable	Units	Type	Range	Trajectory 0	Trajectory 1	Description
Raw A/D input	rad(t)	A/D counts	integer	[0, 4095]	0	rw(t)	Output of A/D converter
Converted weight	cwt(t)	lbs	double	[-1500, 13500]	0	3.663*rw(t) - 1500	Raw weight converted to lbs.

Name	Variable	Units	Type	Range	Trajectory 0	Trajectory 1	Description
Last Axle Timer	lat(t)	sec	integer	[0, tla]	0	$\frac{d}{dt}lat(t) = 1$	Last axle timer
Post-trigger timer	pt(t)	sec	integer	[0, P]	0	$\frac{d}{dt}pt(t) = 1$	Timer to denote end of data after weight signal goes below 300 lbs.
Weight history	wh(j)	lbs	double vector	[-1500, 13500] (each element)	{0, …}	wh(j) = cwt(t(i)), where j = i mod N and where t(i) is the i^{th} time step	Vector of last n values of cwt(t) up to N (=400).
Axle detected	ad(t)	sec	double	> 0	0	ad(t) = last non-zero value of tst defined in Table 5 below.	Time at which an axle was first detected

Table 3: Continuous Output (*Y*)

Name	Variable	Units	Type	Range	Trajectory	Description
Raw A/D	ro(t)	A/D counts	integer	[0, 4095]	rad(t)	Raw A/D counts
Static weight	swt(t)	lbs	double	[-1500, 13500]	cwt(t)	Stream of static weight values sent to host

Notes on continuous signal declarations:

Type and range information are provided for all continuous signals.
One or more trajectories for each controllable continuous signal (the *X* and *Y* sets) are discovered by the HSBS process. In this example, two distinct trajectories were identified for each signal in *X*, while only one was required for each signal in *Y*. In the general case any number of trajectories may be required for any signal in either set.

Table 4: Input Stimuli (*I*)

Name	Variable	Index	Description
null	null	0	Null stimulus

Standby	stb	1	Command to enter standby mode
Static Scale	ssc	2	Command to enter static scale mode
WIM Scale	wsc	3	Command to enter dynamic scale mode
Send raw	srw	4	Command to send raw A/D values
Pad ID	pid	5	Command to send pad ID
Last axle timer restart	atr	6	Restart last axel timer

Table 5: Internal Events (*H*)

Name	Variable	Type	Units	Value When Activated	Description
Calc Start	tst	double	sec	t_0 - t_{pre}, where t_0 is axle start time and t_{pre} is the pretrigger interval	Weight calculation start time
Calc End	tnd	double	sec	t_N, where t_N is axle end time	Weight calculation end time

Table 6: Discrete Responses (*O*)

Name	Variable	Type	Units	Value When Activated	Description
Axle Calculation	iaw	double	lbs	See MatLab function calcs() in Appendix	Axle weight of a tire determined by a convolution integral of the instant the tire touches the pad until it leaves.
	aaw	double	lbs		Axle weight of a tire determined by the average of the weight when the tire is completely on the pad
	std	double	lbs		Standard deviation of weight relative to fit
	tsp	double	ft/sec		Mean tire speed
	toc	double	sec		Time when tire is on center of pad
	gsf	double	lbs		RMS variation of measured speed to low order polynomial fit
	tfl	double	ft		Length of tire footprint on pad
Bad Avg Weight	baw	binary	N/A	1	A/D input = 0 or 4095
Axle timeout	ato	binary	N/A	1	Axle timer timeout

Name	Variable	Type	Units	Value When Activated	Description
Pad ID Msg	pim	integer	N/A	Pad ID number	Sending pad ID to host
Bad Too Fast	btf	binary	N/A	1	WIM Scale mode entered with weight >= 300.

Notes on discrete signal definitions:

All discrete signals are viewed as the values of step functions with a duration of one time step.
Input stimuli are assigned indices to identify them in the Simulink™ implementation.
One and only one input stimulus occurs at each time step. (The default value is 0, indicating the *null* stimulus.)
The default value of signals in *H* and *O* is 0.
One or more of the *H* and *O* signals may be activated (set non-zero) for a time step as specified in the the HSBS.
The Axle Calculation is a vector of values that are assumed to be computed simultaneously, when activated.

EHA Specification

The EHA is derived from the HSBS using the process described below.

For each canonical sequence, assign a mode identifier and do the following:

1. List the applicable autonomous and continuous output signal trajectories from Tables 2 and 3.
2. For each *action* (stimulus-predicate pair) enumerated for the current mode,
    a. Specify the internal events activated from Table 5 and discrete responses from Table 6,
    b. Specify the next mode, and
    c. Specify the initial conditions applicable in the next mode for any autonomous signals defined by differential equations in time.

The results of step 1 are listed in Table 7.

Table 7: WIM DAP Modes and Associated Trajectories

Tag	Mode Index	rad(t)	cwt(t)	$\frac{d}{dt}lat(t)$	$\frac{d}{dt}pt(t)$	ad(t)	wh(j)	ro(t)	swt(t)	Description
0.0.0	0	0	0	0	0	0	{0, …}	0	0	Not active or Standby
1.3.1	1	0	f(rw(t))	0	0	0	{cwt(t(i))}	0	cwt(t)	Static Scale Mode
1.3.2	2	0	0	0	0	0	{cwt(t(i))}	0	0	Bad ADC in Static Scale Model
1.4.1	3	0	f(rw(t))	0	0	0	{cwt(t(i))}	0	0	WIM Scale Mode, No Calc Pending
1.4.2	4	0	0	0	0	0	{cwt(t(i))}	0	0	Bad ADC in WIM Scale Mode
1.5.0	5	u(t)	f(rw(t))	0	0	0	{cwt(t(i))}	rad(t)	0	Send Raw Mode
2.1.2	6	0	f(rw(t))	0	0	tst	{cwt(t(i))}	0	0	Precalc Acqusition
2.7.1	7	0	f(rw(t))	1	0	tst	{cwt(t(i))}	0	0	Precalc Acqusition w/ LAT Counting
2.7.2	8	0	f(rw(t))	1	0	0	{cwt(t(i))}	0	0	LAT Counting w/ No Calc Pending

Tag	Mode Index	rad(t)	cwt(t)	$\frac{d}{dt}lat(t)$	$\frac{d}{dt}pt(t)$	ad(t)	wh(j)	ro(t)	swt(t)	Description
3.1.1	9	0	f(rw(t))	0	1	tst	{cwt(t(i))}	0	0	In Post-Trigger
3.1.2	10	0	f(rw(t))	0	0	tst	{cwt(t(i))}	0	0	Precalc Acquisition w/ LAT Timed Out
3.1.3	11	0	f(rw(t))	0	1	tst	{cwt(t(i))}	0	0	In Post-Trigger w/ LAT Timed Out
3.7.1	12	0	f(rw(t))	1	1	tst	{cwt(t(i))}	0	0	In Post-Trigger w/ LAT Counting

Table 8 summarizes the list of actions enumerated for each mode.

Table 8: Transitions Enumerated for Each Mode

Mode	Stimulus	Predicate	New Mode
0.0.0	null		0.0.0
	stb		0.0.0
	ssc	rad(t) = (0,4095)	1.3.1
		rad(t) ∈ {0, 4095}	1.3.2
	wsc	rad(t) ∈ {0, 4095}	1.4.2
		rad(t) = (0,4095) ∧ cwt(t) = [-1500,300)	1.4.1
		rad(t) = (0,4095) ∧ cwt(t) = [300,13500]	1.3.1
	srw		1.5.0
	pid		0.0.0
	atr		0.0.0
1.3.1	null	rad(t) = (0,4095)	1.3.1
		rad(t) ∈ {0, 4095}	1.3.2
	stb		0.0.0
	ssc		1.3.1
	wsc	rad(t) ∈ {0, 4095}	1.4.2
		rad(t) = (0,4095) ∧ cwt(t) = [-1500,300)	1.4.1
		rad(t) = (0,4095) ∧ cwt(t) = [300,13500]	1.3.1
	srw		1.5.0

Mode	Stimulus	Predicate	New Mode
	pid		1.3.1
	atr		1.3.1
1.3.2	null	rad(t) $\in$ {0, 4095}	1.3.2
		rad(t) = (0,4095)	1.3.1
	stb		0.0.0
	ssc		1.3.2
	wsc	rad(t) $\in$ {0, 4095}	1.4.2
		rad(t) = (0,4095) $\wedge$ cwt(t) = [-1500,300)	1.4.1
		rad(t) = (0,4095) $\wedge$ cwt(t) = [300,13500]	1.3.1
	srw		1.5.0
	pid		1.3.2
	atr		1.3.2
1.4.1	null	rad(t) $\in$ {0, 4095}	1.4.2
		rad(t) = (0,4095) $\wedge$ cwt(t) = [-1500,300)	1.4.1
		rad(t) = (0,4095) $\wedge$ cwt(t) = [300,13500]	2.1.2
	stb		0.0.0
	ssc	rad(t) = (0,4095)	1.3.1
		rad(t) $\in$ {0, 4095}	1.3.2
	wsc		1.4.1
	srw		1.5.0
	pid		1.4.1
	atr	rad(t) $\in$ {0, 4095}	0.0.0
		rad(t) = (0,4095) $\wedge$ lat(t) = (tla, inf)	illegal
		rad(t) = (0,4095) $\wedge$ lat(t)= [0, tla) $\wedge$ cwt(t) = [300,13500]	2.7.1
		rad(t) = (0,4095) $\wedge$ lat(t)= [0, tla) $\wedge$ cwt(t) = [-1500,300)	2.7.2

Mode	Stimulus	Predicate	New Mode
1.4.2	null	rad(t) ∈ {0, 4095}	1.4.2
		rad(t) = (0,4095) Λ cwt(t) = [-1500,300)	1.4.1
		rad(t) = (0,4095) Λ cwt(t) = [300,13500]	1.3.1
	stb		0.0.0
	ssc	rad(t) = (0,4095)	1.3.1
		rad(t) ∈ {0, 4095}	1.3.2
	wsc		1.4.2
	srw		1.5.0
	pid		1.4.2
	atr	rad(t) ∈ {0, 4095}	0.0.0
		rad(t) = (0,4095) Λ lat(t) = [tla, inf)	illegal
		rad(t) = (0,4095) Λ lat(t)= [0, tla) Λ cwt(t) = [300,13500]	1.3.1
		rad(t) = (0,4095) Λ lat(t)= [0, tla) Λ cwt(t) = [-1500,300)	2.7.2
1.5.0	null		1.5.0
	stb		0.0.0
	ssc	rad(t) = (0,4095)	1.3.1
		rad(t) ∈ {0, 4095}	1.3.2
	wsc	rad(t) ∈ {0, 4095}	1.4.2
		rad(t) = (0,4095) Λ cwt(t) = [-1500,300)	1.4.1
		rad(t) = (0,4095) Λ cwt(t) = [300,13500]	1.3.1
	srw		1.5.0
	pid		1.5.0
	atr		1.5.0
2.1.2	null	rad(t) ∈ {0, 4095}	1.4.2
		rad(t) = (0,4095) Λ cwt(t) = [300,13500]	2.1.2

Mode	Stimulus	Predicate	New Mode
		rad(t) = (0,4095) Λ cwt(t) = [-1500,300) Λ pt(t) = [0, P)	3.1.1
		rad(t) = (0,4095) Λ cwt(t) = [-1500,300) Λ pt(t) = [P, inf)	illegal
	stb		0.0.0
	ssc	rad(t) = (0,4095)	1.3.1
		rad(t) ∈ {0, 4095}	1.3.2
	wsc		2.1.2
	srw		1.5.0
	pid		2.1.2
	atr	rad(t) ∈ {0, 4095}	0.0.0
		rad(t) = (0,4095) Λ lat(t) = [tla, inf)	illegal
		rad(t) = (0,4095) Λ lat(t) = [0, tla) Λ cwt(t) = [300,13500]	2.7.1
		rad(t) = (0,4095) Λ lat(t) = [0, tla) Λ cwt(t) = [-1500,300) Λ pt(t) = [0, P)	3.7.1
		rad(t) = (0,4095) Λ lat(t) = [0, tla) Λ cwt(t) = [-1500,300) Λ pt(t) = [P, inf)	2.7.2
2.7.1	null	rad(t) ∈ {0, 4095}	0.0.0
		rad(t) = (0,4095) Λ cwt(t) = [300,13500] Λ lat(t) = [0, tla)	2.7.1
		rad(t) = (0,4095) Λ cwt(t) = [300,13500] Λ lat(t) = [tla, inf]	3.1.2
		rad(t) = (0,4095) Λ cwt(t) = [-1500,300) Λ lat(t) = [0, tla) Λ pt(t) = [0, P)	3.7.1
		rad(t) = (0,4095) Λ cwt(t) = [-1500,300) Λ lat(t) = [0, tla) Λ pt(t) = [P, inf]	illegal
		rad(t) = (0,4095) Λ cwt(t) = [-1500,300) Λ lat(t) = [tla, inf] Λ pt(t) = [P, inf]	illegal
		rad(t) = (0,4095) Λ cwt(t) = [-1500,300) Λ lat(t) = [tla, inf] Λ pt(t) = [0, P)	3.1.3
	stb		0.0.0
	ssc	rad(t) = (0,4095)	1.3.1
		rad(t) ∈ {0, 4095}	1.3.2
	wsc		2.7.1
	srw		1.5.0

Mode	Stimulus	Predicate	New Mode
	pid		2.7.1
	atr	rad(t) ∈ {0, 4095}	0.0.0
		rad(t) = (0,4095) Λ lat(t) = (tla, inf)	illegal
		rad(t) = (0,4095) Λ lat(t)= [0, tla) Λ cwt(t) = [300,13500]	2.7.1
		rad(t) = (0,4095) Λ lat(t) = [0, tla) Λ cwt(t) = [-1500,300) Λ pt(t) = [0, P)	3.7.1
		rad(t) = (0,4095) Λ lat(t) = [0, tla) Λ cwt(t) = [-1500,300) Λ pt(t) = [P, inf)	illegal
2.7.2	null	rad(t) ∈ {0, 4095}	0.0.0
		rad(t) = (0,4095) Λ cwt(t) = [300,13500] Λ lat(t) = [0, tla)	2.7.1
		rad(t) = (0,4095) Λ cwt(t) = [300,13500] Λ lat(t) = [tla, inf]	3.1.2
		rad(t) = (0,4095) Λ cwt(t) = [-1500,300) Λ lat(t) = [0, tla)	2.7.2
		rad(t) = (0,4095) Λ cwt(t) = [-1500,300) Λ lat(t) = [tla, inf]	0.0.0
	stb		0.0.0
	ssc	rad(t) = (0,4095)	1.3.1
		rad(t) ∈ {0, 4095}	1.3.2
	wsc		2.7.2
	srw		1.5.0
	pid		2.7.2
	atr	rad(t) ∈ {0, 4095}	0.0.0
		rad(t) = (0,4095) Λ cwt(t) = [300,13500] Λ lat(t) = [0, tla)	2.7.1
		rad(t) = (0,4095) Λ cwt(t) = [300,13500] Λ lat(t) = [tla, inf]	illegal
		rad(t) = (0,4095) Λ cwt(t) = [-1500,300) Λ lat(t) = [0, tla)	2.7.2
		rad(t) = (0,4095) Λ cwt(t) = [-1500,300) Λ lat(t) = [tla, inf]	illegal
3.1.1	null	rad(t) = {0, 4095}	1.4.2
		rad(t) = (0,4095) Λ cwt(t) = [300,13500]	2.1.2
		rad(t) = (0,4095) Λ cwt(t) = [-1500,300) Λ pt(t) = [0, P)	3.1.1

Mode	Stimulus	Predicate	New Mode
		rad(t) = (0,4095) Λ cwt(t) = [-1500,300) Λ pt(t) = [P, inf)	1.4.1
	stb		0.0.0
	ssc	rad(t) = (0,4095)	1.3.1
		rad(t) ∈ {0, 4095}	1.3.2
	wsc		3.1.1
	srw		1.5.0
	pid		3.1.1
	atr	rad(t) ∈ {0, 4095}	0.0.0
		rad(t) = (0,4095) Λ lat(t) = (tla, inf]	illegal
		rad(t) = (0,4095) Λ lat(t)= [0, tla) Λ cwt(t) = [300,13500]	2.7.1
		rad(t) = (0,4095) Λ lat(t)= [0, tla) Λ cwt(t) = [-1500,300) Λ pt(t) = [0, P)	3.7.1
		rad(t) = (0,4095) Λ lat(t)= [0, tla) Λ cwt(t) = [-1500,300) Λ pt(t) = [P, inf)	2.7.2
3.7.1	null	rad(t) ∈ {0, 4095}	0.0.0
		rad(t) = (0,4095) Λ cwt(t) = [300,13500] Λ lat(t) = [0, tla)	2.7.1
		rad(t) = (0,4095) Λ cwt(t) = [300,13500] Λ lat(t) = [tla, inf]	3.1.2
		rad(t) = (0,4095) Λ cwt(t) = [-1500,300) Λ lat(t) = [0, tla) Λ pt(t) = [0, P)	3.7.1
		rad(t) = (0,4095) Λ cwt(t) = [-1500,300) Λ lat(t) = [0, tla) Λ pt(t) = [P, inf]	2.7.2
		rad(t) = (0,4095) Λ cwt(t) = [-1500,300) Λ lat(t) = [tla, inf] Λ pt(t) = [0, P)	3.1.3
		rad(t) = (0,4095) Λ cwt(t) = [-1500,300) Λ lat(t) = [tla, inf] Λ pt(t) = [P, inf]	0.0.0
	stb		0.0.0
	ssc	rad(t) = (0,4095)	1.3.1
		rad(t) ∈ {0, 4095}	1.3.2
	wsc		3.7.1
	srw		1.5.0
	pid		3.7.1

Mode	Stimulus	Predicate	New Mode
	atr	rad(t) ∈ {0, 4095}	0.0.0
		rad(t) = (0,4095) Λ lat(t) = (tla, inf]	illegal
		rad(t) = (0,4095) Λ lat(t)= [0, tla) Λ cwt(t) = [300,13500]	3.1.2
		rad(t) = (0,4095) Λ lat(t)= [0, tla) Λ cwt(t) = [-1500,300) Λ pt(t) = [0, P)	3.7.1
		rad(t) = (0,4095) Λ lat(t)= [0, tla) Λ cwt(t) = [-1500,300) Λ pt(t) = [P, inf)	2.7.2
3.1.2	null	rad(t) ∈ {0, 4095}	1.4.2
		rad(t) = (0,4095) Λ cwt(t) = [300,13500]	3.1.2
		rad(t) = (0,4095) Λ cwt(t) = [-1500,300) Λ pt(t) = [0, P)	3.1.3
		rad(t) = (0,4095) Λ cwt(t) = [-1500,300) Λ pt(t) = [P, inf)	illegal
	stb		0.0.0
	ssc	rad(t) ∈ (0,4095)	1.3.1
		rad(t) ∈ {0, 4095}	1.3.2
	wsc		3.1.2
	srw		1.5.0
	pid		3.1.2
	atr	rad(t) ∈ {0, 4095}	0.0.0
		rad(t) = (0,4095) Λ lat(t) = [tla, inf]	illegal
		rad(t) = (0,4095) Λ lat(t)= [0, tla) Λ cwt(t) = [300,13500]	2.7.1
		rad(t) = (0,4095) Λ lat(t)= [0, tla) Λ cwt(t) = [-1500,300) Λ pt(t) = [0, P)	3.7.1
		rad(t) = (0,4095) Λ lat(t)= [0, tla) Λ cwt(t) = [-1500,300) Λ pt(t) = [P, inf)	illegal
3.1.3	null	rad(t) ∈ {0, 4095}	0.0.0
		rad(t) = (0,4095) Λ cwt(t) = [300,13500]	2.1.2
		rad(t) = (0,4095) Λ cwt(t) = [-1500,300) Λ pt(t) = [0, P)	3.1.3
		rad(t) = (0,4095) Λ cwt(t) = [-1500,300) Λ pt(t) = [P, inf)	1.4.1
	stb		0.0.0

Mode	Stimulus	Predicate	New Mode
	ssc	rad(t) = (0,4095)	1.3.1
		rad(t) ∈ {0, 4095}	1.3.2
	wsc		3.1.3
	srw		1.5.0
	pid		3.1.3
	atr	rad(t) ∈ {0, 4095}	0.0.0
		rad(t) = (0,4095) Λ lat(t) = (tla, inf]	illegal
		rad(t) = (0,4095) Λ lat(t)= [0, tla) Λ cwt(t) = [300,13500]	2.7.1
		rad(t) = (0,4095) Λ lat(t)= [0, tla) Λ cwt(t) = [-1500,300) Λ pt(t) = [0, P)	3.7.1
		rad(t) = (0,4095) Λ lat(t)= [0, tla) Λ cwt(t) = [-1500,300) Λ pt(t) = [P, inf)	2.7.2

To facilitate Simulink™ implementation, the results of steps 2a through 2c are presented inTables 9 through 21 as a series of *condition vector* matrices, each applicable to a respective **Mode** in Table 8. Each column corresponds to a transition from an initial **Mode** (named in the caption) to a specified **New Mode (**given in the row labeled **mode)**.

Each column is a vector of indices. The first index in each column is the zero-based index of the destination mode. The remaining indices in each column are used for selection of specific condition vector values from Table 22. The rows labeled **tst** through **btf** specify the activation status of the internal events and discrete responses defined in Tables 5 and 6. The indices in the rows labeled **lat(0)** and **pt(0)** specify the initial conditions of those respective signals on entry to the new mode. Note that the Axle Calculation variables are treated as a vector represented by the **calcs** rows below.

Table 9: Condition Vectors for Transitions from Mode 0

mode	0	0	1	2	4	3	1	5	0	0
tst	0	0	0	0	0	0	0	0	0	0
tnd	0	0	0	0	0	0	0	0	0	0
calc	0	0	0	0	0	0	0	0	0	0
baw	0	0	0	1	1	0	0	0	0	0
ato	0	0	0	0	0	0	0	0	0	0
pim	0	0	0	0	0	0	0	0	1	0
btf	0	0	0	0	0	0	1	0	0	0
lat(0)	0	0	0	0	0	0	0	0	0	0
pt(0)	0	0	0	0	0	0	0	0	0	0

Table 10: Condition Vectors for Transitions from Mode 1

mode	1	2	0	1	4	3	1	5	1	1
tst	0	0	0	0	0	0	0	0	0	0

tnd	0	0	0	0	0	0	0	0	0	0
calc	0	0	0	0	0	0	0	0	0	0
baw	0	1	0	0	1	0	0	0	0	0
ato	0	0	0	0	0	0	0	0	0	0
pim	0	0	0	0	0	0	0	0	1	0
btf	0	0	0	0	0	0	1	0	0	0
lat(0)	0	0	0	0	0	0	0	0	0	0
pt(0)	0	0	0	0	0	0	0	0	0	0

Table 11: Condition Vectors for Transitions from Mode 2

mode	2	2	0	2	4	3	1	5	2	2
tst	0	0	0	0	0	0	0	0	0	0
tnd	0	0	0	0	0	0	0	0	0	0
calc	0	0	0	0	0	0	0	0	0	0
baw	0	0	0	0	1	0	0	0	0	0
ato	0	0	0	0	0	0	0	0	0	0
pim	0	0	0	0	0	0	0	0	1	0
btf	0	0	0	0	0	0	1	0	0	0
lat(0)	0	0	0	0	0	0	0	0	0	0
pt(0)	0	0	0	0	0	0	0	0	0	0

Table 12: Condition Vectors for Transitions from Mode 3

mode	4	3	6	0	1	2	3	5	3	0	Illegal	7	8
tst	0	0	1	0	0	0	0	0	0	0		1	0
tnd	0	0	0	0	0	0	0	0	0	0		0	0
calc	0	0	0	0	0	0	0	0	0	0		0	0
baw	1	0	0	0	0	1	0	0	0	1		0	0
ato	0	0	0	0	0	0	0	0	0	0		0	0
pim	0	0	0	0	0	0	0	0	1	0		0	0
btf	0	0	0	0	0	0	0	0	0	0		0	0
lat(0)	0	0	0	0	0	0	0	0	0	0		1	1
pt(0)	0	0	0	0	0	0	0	0	0	0		0	0

Table 13: Condition Vectors for Transitions from Mode 4

mode	4	3	6	0	1	2	4	5	4	0	Illegal	7	8
tst	0	0	1	0	0	0	0	0	0	0		0	0
tnd	0	0	0	0	0	0	0	0	0	0		0	0
calc	0	0	0	0	0	0	0	0	0	0		0	0
baw	0	0	0	0	0	0	0	0	0	0		0	0
ato	0	0	0	0	0	0	0	0	0	0		0	0
pim	0	0	0	0	0	0	0	0	1	0		0	0
btf	0	0	0	0	0	0	0	0	0	0		0	0
lat(0)	0	0	0	0	0	0	0	0	0	0		1	1
pt(0)	0	0	0	0	0	0	0	0	0	0		0	0

Table 14: Condition Vectors for Transitions from Mode 5

mode	5	0	1	2	4	3	1	5	5	5
tst	0	0	0	0	0	0	0	0	0	0
tnd	0	0	0	0	0	0	0	0	0	0
calc	0	0	0	0	0	0	0	0	0	0
baw	0	0	0	1	1	0	0	0	0	0
ato	0	0	0	0	0	0	0	0	0	0
pim	0	0	0	0	0	0	0	0	1	0
btf	0	0	0	0	0	0	1	0	0	0
lat(0)	0	0	0	0	0	0	0	0	0	0
pt(0)	0	0	0	0	0	0	0	0	0	0

Table 15: Condition Vectors for Transitions from Mode 6

mode	4	6	9	illegal	0	1	2	6	5	6	0	illegal	7	12	8
tst	0	0	0		0	0	0	0	0	0	0		0	0	0
tnd	0	0	0		0	0	0	0	0	0	0		0	0	1
calc	0	0	0		0	0	0	0	0	0	0		0	0	1
baw	1	0	0		0	0	1	0	0	0	1		0	0	0
ato	0	0	0		0	0	0	0	0	0	0		0	0	0
pim	0	0	0		0	0	0	0	0	1	0		0	0	0
btf	0	0	0		0	0	0	0	0	0	0		0	0	1
lat(0)	0	0	0		0	0	0	0	0	0	0		0	0	0
pt(0)	0	0	1		0	0	0	0	0	0	0		0	1	0

Table 16: Condition Vectors for Transitions from Mode 7

mode	0	7	10	12	illegal	illegal	11	0	1	2	7	5	7	0	illegal	7	12	illegal
tst	0	0	0	0			0	0	0	0	0	0	0	0		0	0	
tnd	0	0	0	0			0	0	0	0	0	0	0	0		0	0	
calc	0	0	0	0			0	0	0	0	0	0	0	0		0	0	
baw	1	0	0	0			0	0	0	1	0	0	0	1		0	0	
ato	0	0	0	0			1	0	0	0	0	0	0	0		0	0	
pim	0	0	0	0			0	0	0	0	0	0	1	0		0	0	
btf	0	0	0	0			0	0	0	0	0	0	0	0		0	0	
lat(0)	0	1	0	1			0	0	0	0	0	0	0	0		1	0	
pt(0)	0	0	0	0			0	0	0	0	0	0	0	0		0	0	

Table 17: Condition Vectors for Transitions from Mode 8

mode	0	7	10	8	0	0	1	2	8	5	8	0	7	illegal	8	illegal
tst	0	0	0	0	0	0	0	0	0	0	0	0	0		0	
tnd	0	0	0	0	0	0	0	0	0	0	0	0	0		0	
calc	0	0	0	0	0	0	0	0	0	0	0	0	0		0	
baw	1	0	0	0	0	0	0	1	0	0	0	1	0		0	
ato	0	0	1	0	1	0	0	0	0	0	0	0	0		0	
pim	0	0	0	0	0	0	0	0	0	0	1	0	0		0	
btf	0	0	0	0	0	0	0	0	0	0	0	0	0		0	

lat(0)	0	1	0	1	0	0	0	0	0	0	0	0	1		0	
pt(0)	0	0	0	0	0	0	0	0	0	0	0	0	0		0	

Table 18: Condition Vectors for Transitions from Mode 9

mode	4	6	9	3	0	1	2	9	5	9	0	illegal	7	12	8
tst	0	0	0	0	0	0	0	0	0	0	0		0	0	0
tnd	1	1	0	1	1	0	1	0	0	0	1		1	0	1
calc	1	1	0	1	1	0	1	0	0	0	1		1	0	1
baw	1	0	0	0	0	0	1	0	0	0	1		0	0	0
ato	0	0	0	0	0	0	0	0	0	0	0		0	0	0
pim	0	0	0	0	0	0	0	0	0	1	0		0	0	0
btf	0	0	0	0	0	0	0	0	0	0	0		0	0	0
lat(0)	0	0	0	0	0	0	0	0	0	0	0		0	0	0
pt(0)	0	0	1	0	0	0	0	1	0	1	0		0	1	0

Table 19: Condition Vectors for Transitions from Mode 10

mode	0	7	10	12	8	11	0	0	1	2	12	5	12	0	il	10	12	8
tst	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0
tnd	0	1	1	0	1	0	1	1	1	1	0	1	0	1		1	0	1
calc	0	1	1	0	1	0	1	1	1	1	0	1	0	1		1	0	1
baw	1	0	0	0	0	0	0	0	0	1	0	0	0	1		0	0	0
ato	0	0	1	0	0	1	1	0	0	0	0	0	0	0		0	0	0
pim	0	0	0	0	0	0	0	0	0	0	0	0	1	0		0	0	0
btf	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0
lat(0)	0	1	0	1	1	0	0	0	0	0	1	0	1	0		0	0	0
pt(0)	0	0	0	1	0	1	0	0	0	0	1	0	1	0		0	1	1

Table 20: Condition Vectors for Transitions from Mode 11

mode	4	10	11	illegal	0	1	2	10	5	10	0	illegal	7	12	illegal
tst	0	0	0		0	0	0	0	0	0	0		0	0	
tnd	0	0	0		0	0	0	0	0	0	0		0	0	
calc	0	0	0		0	0	0	0	0	0	0		0	0	
baw	1	0	0		0	0	1	0	0	0	1		0	0	
ato	0	0	0		0	0	0	0	0	0	0		0	0	
pim	0	0	0		0	0	0	0	0	1	0		0	0	
btf	0	0	0		0	0	0	0	0	0	0		0	0	
lat(0)	0	0	0		0	0	0	1	0	1	0		0	0	
pt(0)	0	0	0		0	0	0	0	0	0	0		0	0	

Table 21: Condition Vectors for Transitions from Mode 12

mode	0	6	11	3	0	1	2	11	5	11	0	illegal	7	12	8
tst	0	0	0	0	0	0	0	0	0	0	0		0	0	0
tnd	1	1	0	1	1	1	1	0	1	0	1		1	0	1
calc	1	1	0	1	1	1	1	0	1	0	1		1	0	1
baw	1	0	0	0	0	0	1	0	0	0	1		0	0	0

ato	0	0	0	0	0	0	0	0	0	0	0		0	0	0
pim	0	0	0	0	0	0	0	0	0	1	0		0	0	0
btf	0	0	0	0	0	0	0	0	0	0	0		0	0	0
lat(0)	0	0	0	0	0	0	0	0	0	0	0		0	0	0
pt(0)	0	0	1	0	0	0	0	1	0	1	0		1	0	1

Table 22: Condition Vector Elements

Signal	Declaration	Index = 0	Index = 1
tst	Table 5	0	t - pretrigger
tnd	Table 5	0	t+posttrigger
calc	Table 6 (Axle calculations)	null	Perform axle calculations in Table 6
baw	Table 6	0	1
ato	Table 6	0	1
pim	Table 6	null	pad ID
btf	Table 6	0	1
lat(0)	Table 2	0	lat(last)
pt(0)	Table 2	0	pt(last)

Implementing the EHA as a Simulink™ Model

Figure 1 shows the high level structure of the Simulink™ model.

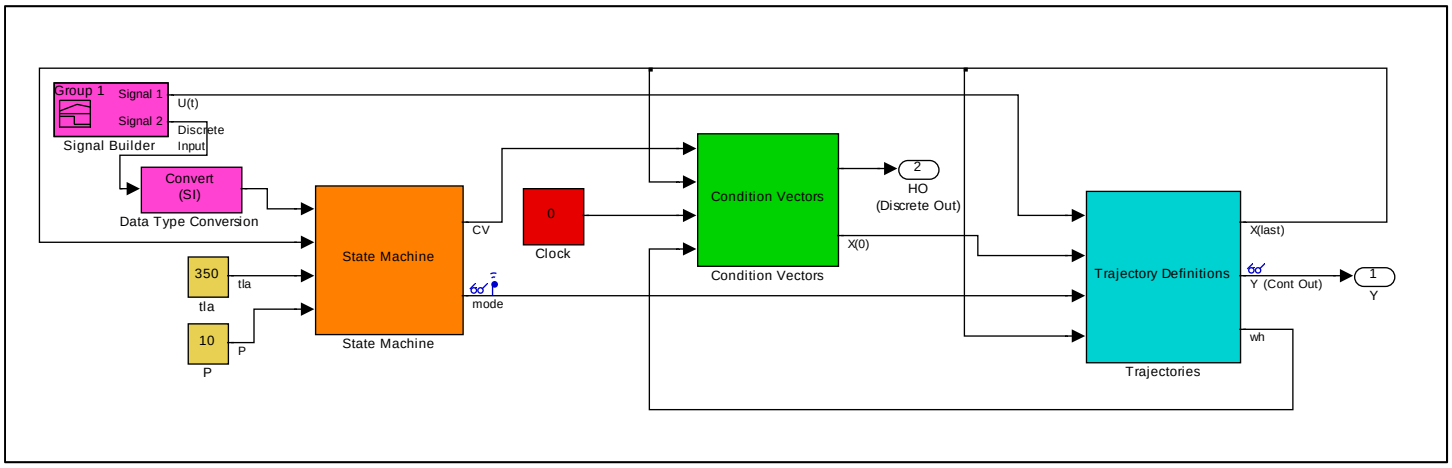

Figure 1. WIM DAP High-level Model Structure

The model has three primary subsystems:

1. State Machine – implements transitions among modes and condition vector selection logic
2. Condition Vectors – implements generation of signals for condition vectors selected by the State Machine
3. Trajectory Definitions – generates continuous signal trajectories for the current mode

Other high level blocks include

1. A Signal Builder for generating continuous and discrete test inputs for the signals defined in Tables 1 and 4.
2. A Type Conversion block to ensure that the discrete inputs are represented as short integers.
3. Constant blocks for the axle and post trigger timer timeout values.
4. A Clock source for use in Condition Vector computations.

The Sink block labeled “Y” receives the vector of continuous output signals (Table 3). The Sink block labeled “HO” receives the vector of internal events (Table 5) and discrete output signals (Table 6).

State Machine

Figure 2 is an overview of the StateMachine subsystem. The overall function of this subsystem is to select a transition and condition vector, based on the current mode, current discrete input, and the current set of codomains for continuous signals. For each mode in the EHA, there are two blocks:

1. A subsystem to determine whether the mode is current and, if so, which transition should occur next.
2. A lookup table to select the new mode and condition vector produced by the selected transition.

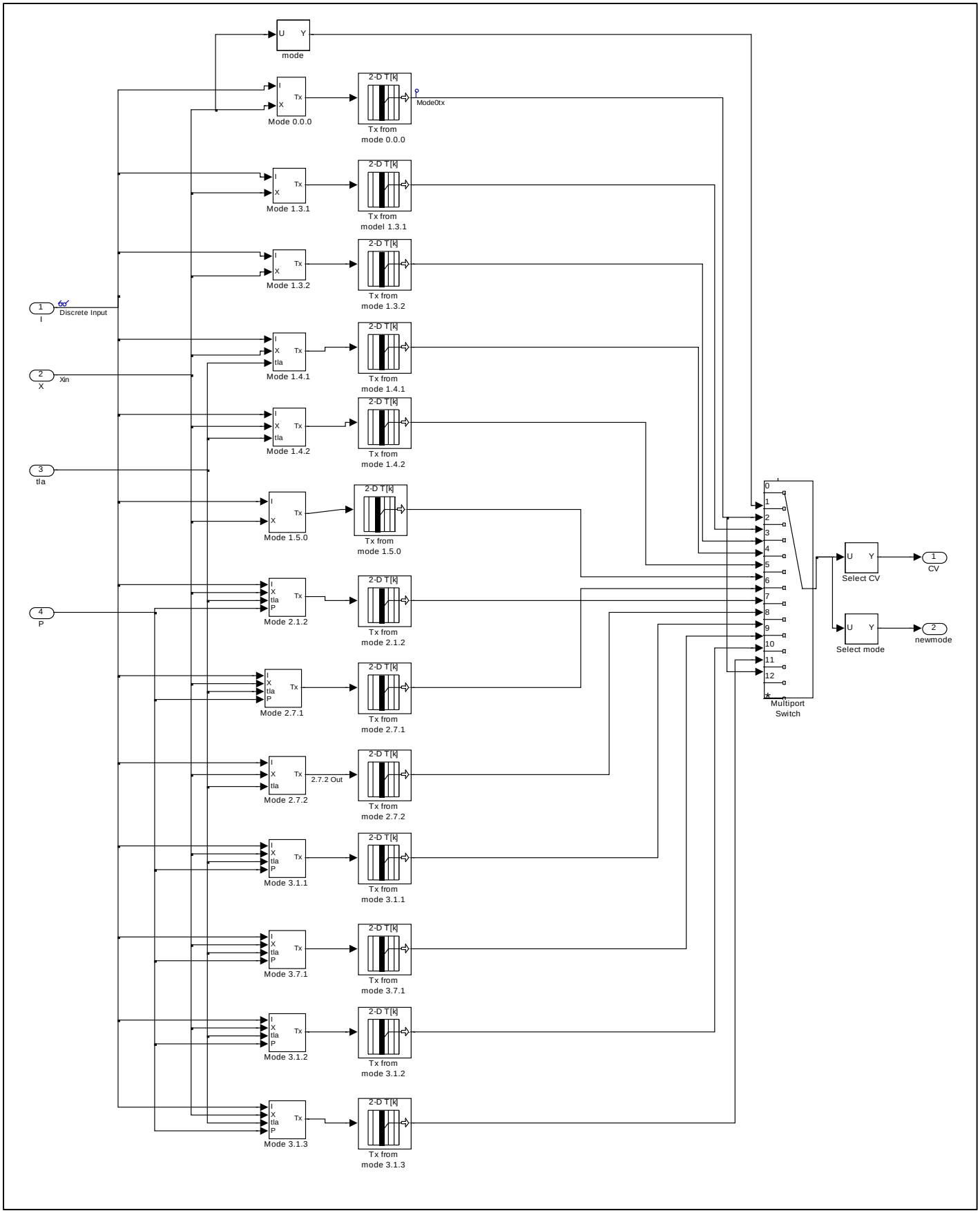

Figure 2: State Machine Subsystem

The transition selection subsystem for a particular mode is implemented as a two-level subsystem hierarchy. At the first level, exemplified in Figure 3, the relevant input signals are applied to individual subsystems for each potential transition. The purpose of each transition subsystem is to determine whether the current discrete input and continuous codomain combination matches a particular stimulus/predicate pair in Table 8. If the stimulus/predicate pair matches, the transition subsystem outputs an index that identifies the transition uniquely for the current mode. Otherwise the transition subsystem output is zero.

As an example, Figure 4 shows the implementation details for subsystem Tx 14 in Figure 3. The conditions necessary for Tx 14 to be active are determined by checking for

- Current stimulus = 6
- Current mode index = 9
- rad(t) $\in$ (0, 4095)
- lat(t) $\in$ [0, tla)
- cwt(t) $\in$ [-1500, 300)
- pt(t) $\in$ [0, P)

Boolean outputs from the blocks implementing these comparisons are ANDed, and the result (1 or 0) is multiplied by the transition index to produce the subsystem output.

Outputs from all the transition subsystems are fed to a multiplexer and then summed (Figure 3). The hybrid enumeration process that produced Table 8 guarantees that only one transition subsystem has a non-zero output for the current mode. Therefore the sum in Figure 3 is either zero or the index of the next transition. The resulting index is decremented to produce a zero-based index to the downstream lookup table.

As shown in Figure 2, the output of each mode subsystem is an index into a lookup table. The lookup tables for the 13 WIM DAP modes are implementations of Tables 9 through 21, respectively. Each transition index selects a column in the associated table. Each column is a list of indices. The first element in each column is the new mode index. Each remaining index selects a condition vector element from Table 22.

Thus, the output of each lookup table is a vector with the new mode as the first element and the condition vector indices as the remaining elements. All lookup table output vectors are input to a multiport switch. The current mode is fed to the control port of the multiport switch to select the output condition vector for the current transition (Figure 2). The new mode and the condition vector are treated as separate outputs from the state machine subsystem.

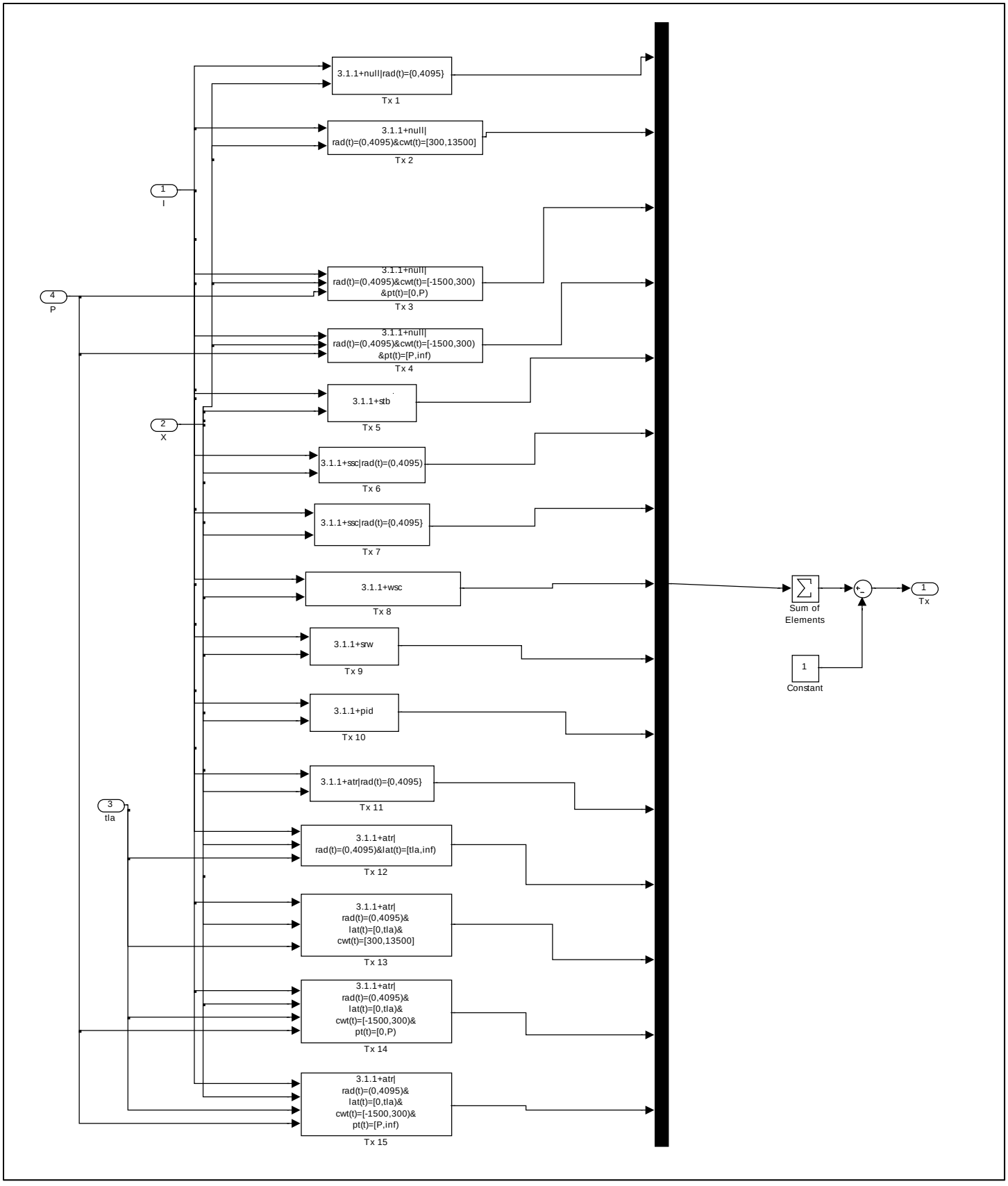

Figure 3: Transition selection for mode 3.1.1

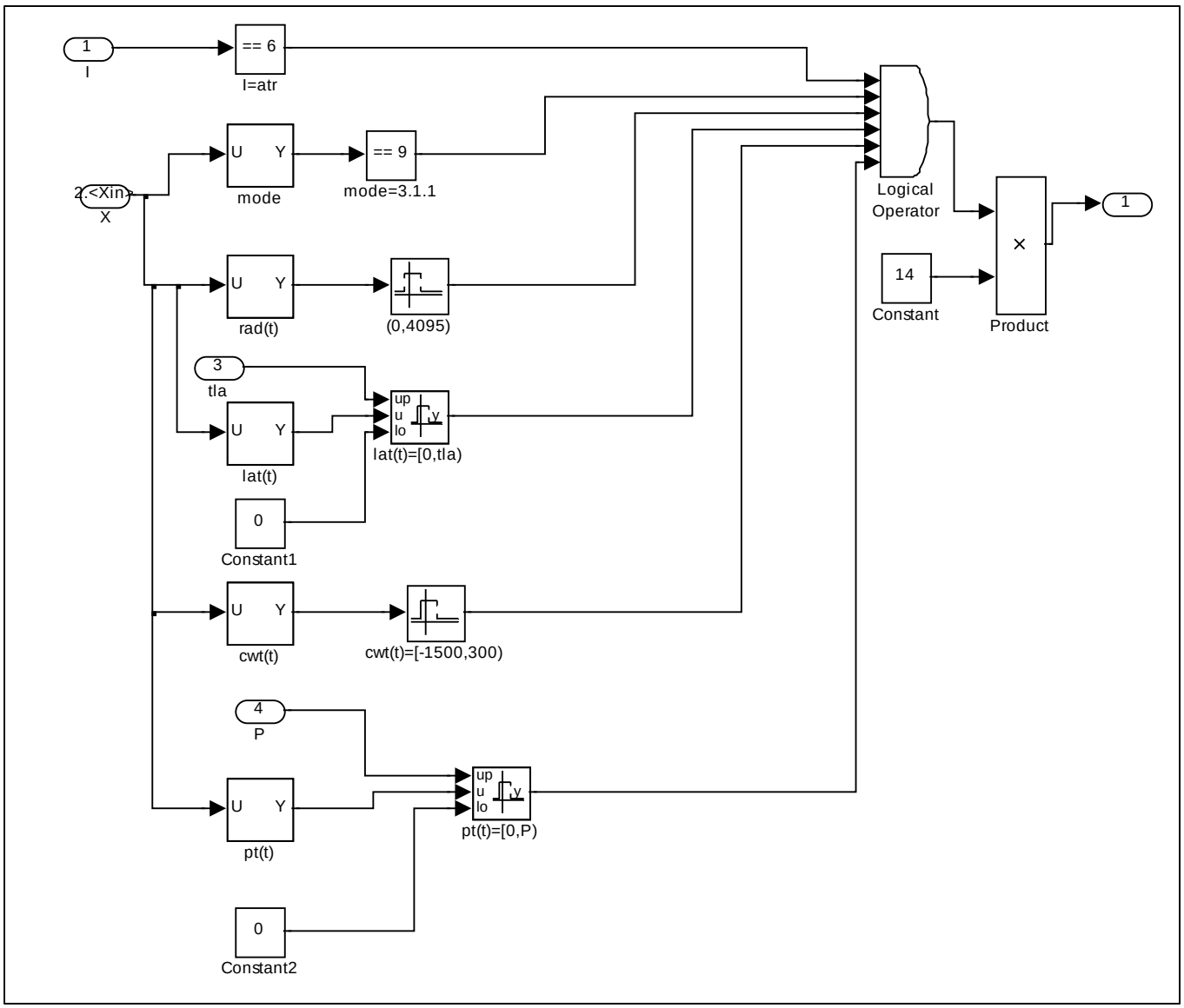

Figure 4: Detailed implementation of TX 14 for mode 3.1.1

Condition Vectors

Figure 5 shows additional detail for the Condition Vectors subsystem. This subsystem generates discrete internal events, discrete outputs, and initial conditions caused by the most recent transition. The general approach is to select a value or computation based on the condition vector indices from the State Machine subsystem. In addition to the condition vector indices, inputs include the vector, *X*, of autonomous continuous signals and the clock signal, *t*. Although the weight history buffer, wh(j), is conceptually part of *X*, Simulink requires it to be handled separately because its sampling mechanism is different from the other signals in *X*.

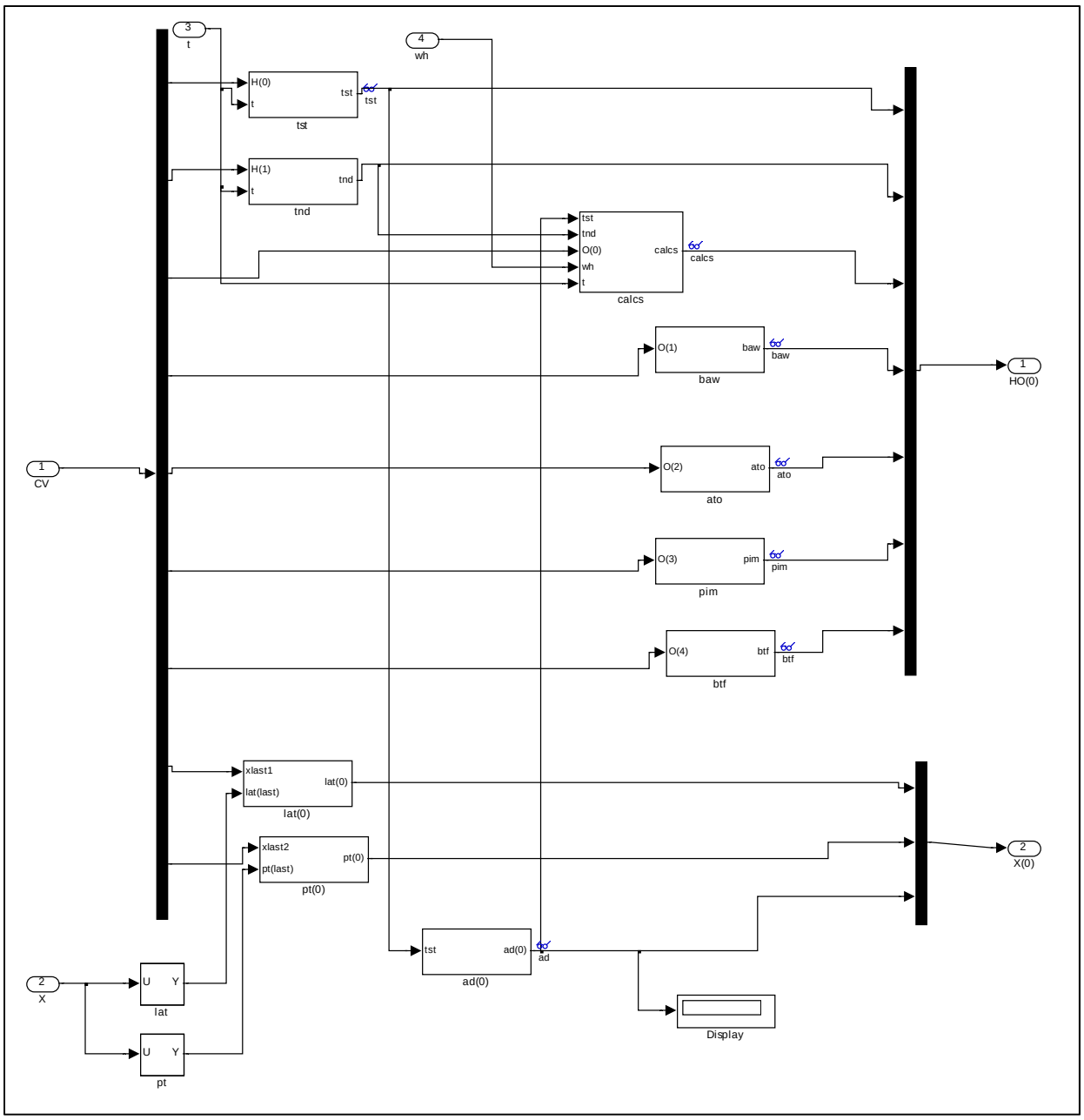

Figure 5: Condition Vectors Overview

Generation of condition vector values for WIM DAP can be described in terms of three implementation patterns:

1. Triggered events
2. Triggered axle calculations
3. Enabled initial conditions

The discrete internal events (tst, tnd) and discrete outputs (baw, ato, pim, btf) are implemented as triggered events. The implementation of baw, shown in Figure 6, is typical. The baw index determined in the State machine subsystem is compared to 0 and 1. If the index is 0, the output value 0 is enabled. If the index is 1, the value 1 is enabled for one time step. The duration is limited to one time step by specifying that the output is triggered on the rising edge of the index change and specifying “Output when disabled:” as “reset” in the sink block of the baw = 1 subsystem.

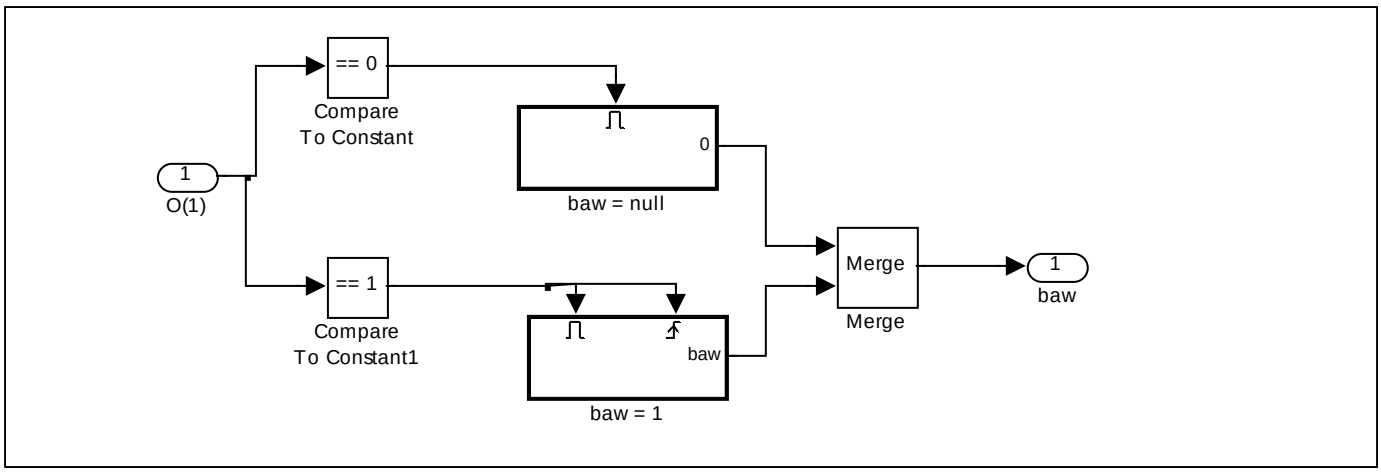

Figure 6: Implementation of baw event

The axle calculations in Table 6 must be performed when a complete weight history data set (wh(j)) has been acquired. In addition to wh(j), the calculations require the current clock value and the most recent values of tst and tnd as inputs. The most recent tst value must be captured via a "Sample and Hold" block in the subsystem labeled ad(0) (see Figure 7). No additiona logic is required to capture the latest tnd value, since it is triggered at the same time as the axle calculations.

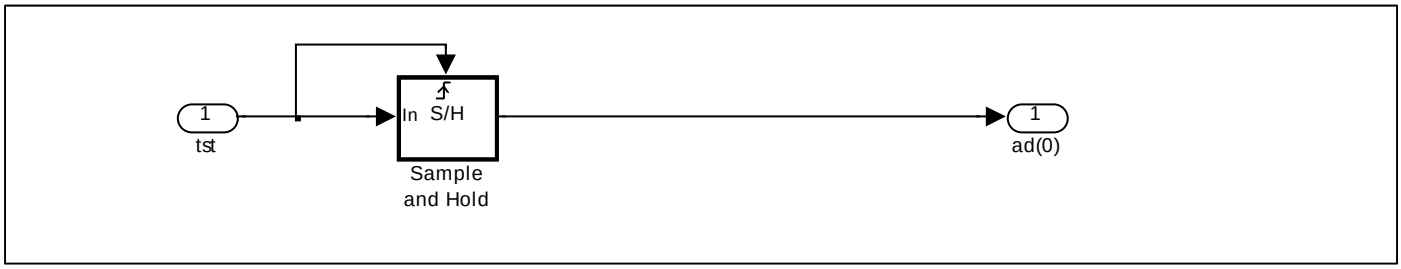

Figure 7: Sample and Hold implementation for tst

As shown in Figure 8, the triggered axle calculations are implemented in a similar manner to the triggered events. If the **calcs** index (O(0) in the figure) determined in the State Machine subsystem is 0, the output of the **calcs** subsystem is a scalar 0. If the **calcs** index is 1, the **compute** subsystem is triggered on the rising edge of the index input. The **compute** subsystem (Figure 9) contains a MatLab™ function block to compute the axle properties listed in Table 6 as a vector. A single MatLab™ function block is used to compute all seven properties to accommodate dependencies among the properties. The **calcs** MatLab™ code is listed in the Appendix.

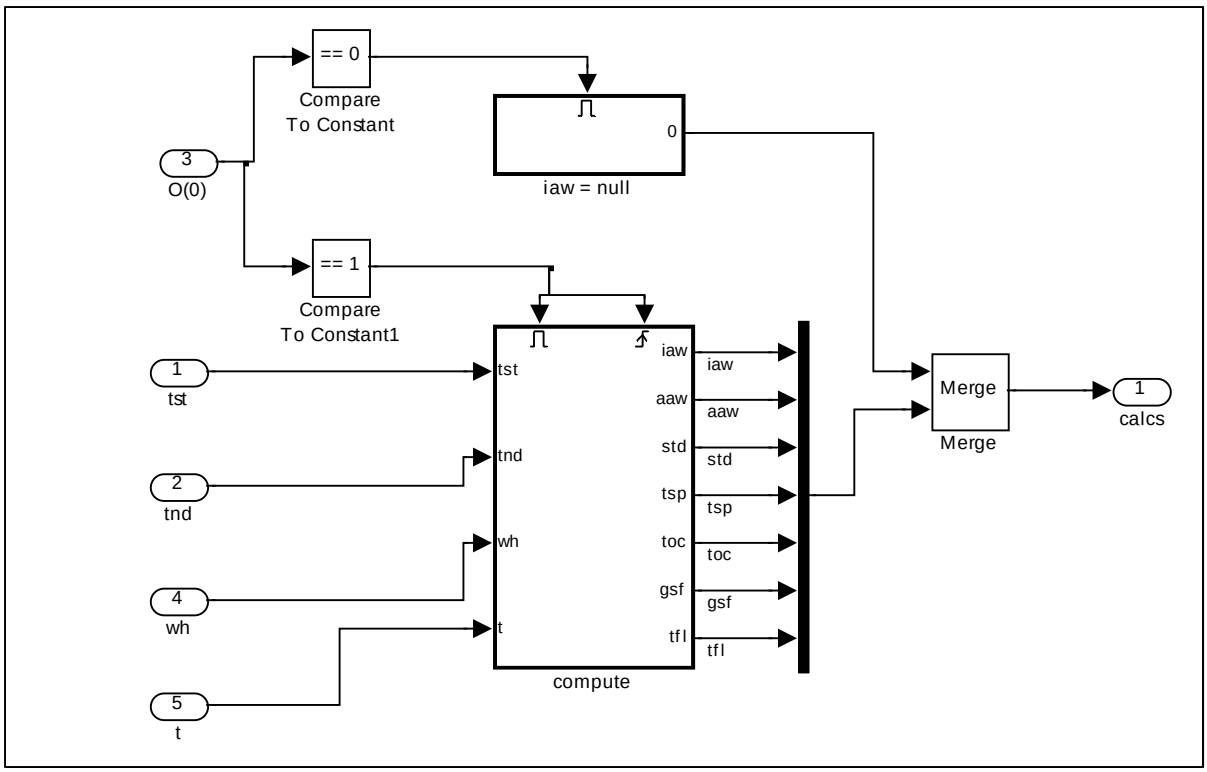

Figure 8: Implementation of triggered axle calculations

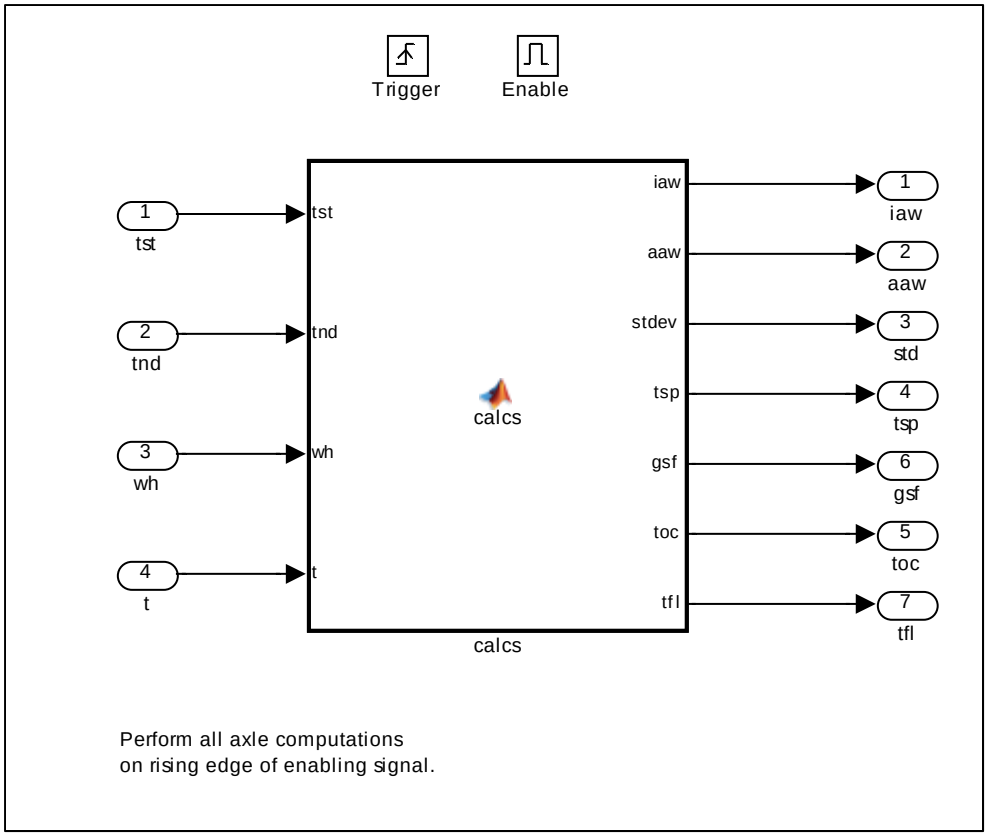

Figure 9: Compute subsystem

The Last Axle Timer, lat(t), and Post-Trigger Timer, pt(t), are defined by differential equations in time and therefore must have initial conditions defined for each mode. The implementation pattern is shown in Figure 10. The applicable initial value is set at each time step, based on the associated index determined by the State machine subsystem.

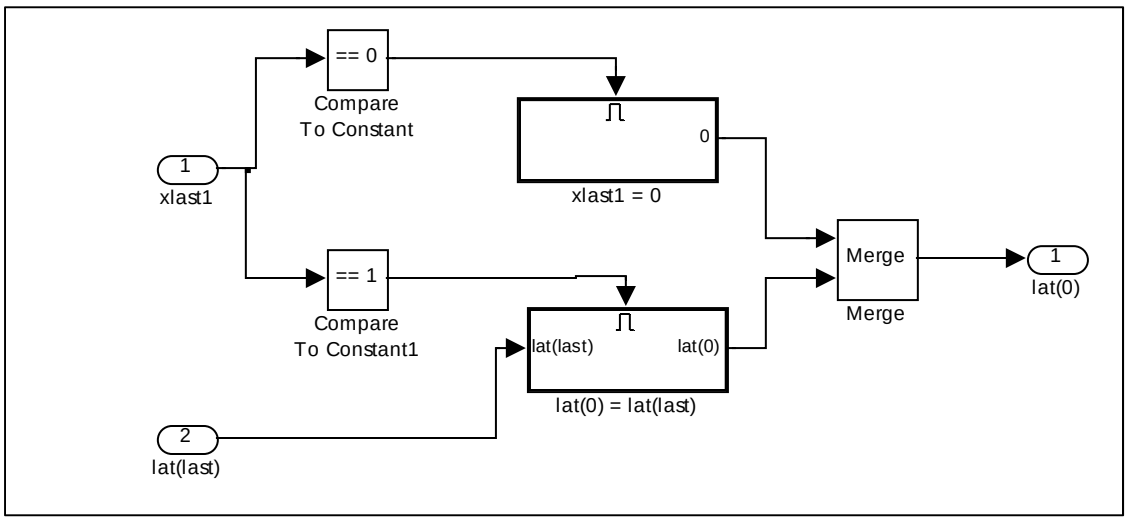

Figure 10: Initial condition selection implementation pattern

Trajectory Definitions

The Trajectory Definitions subsystem selects the set of continuous signal trajectories for the current mode and generates the selected signals. Figure 11 shows an overview of the system implementation.

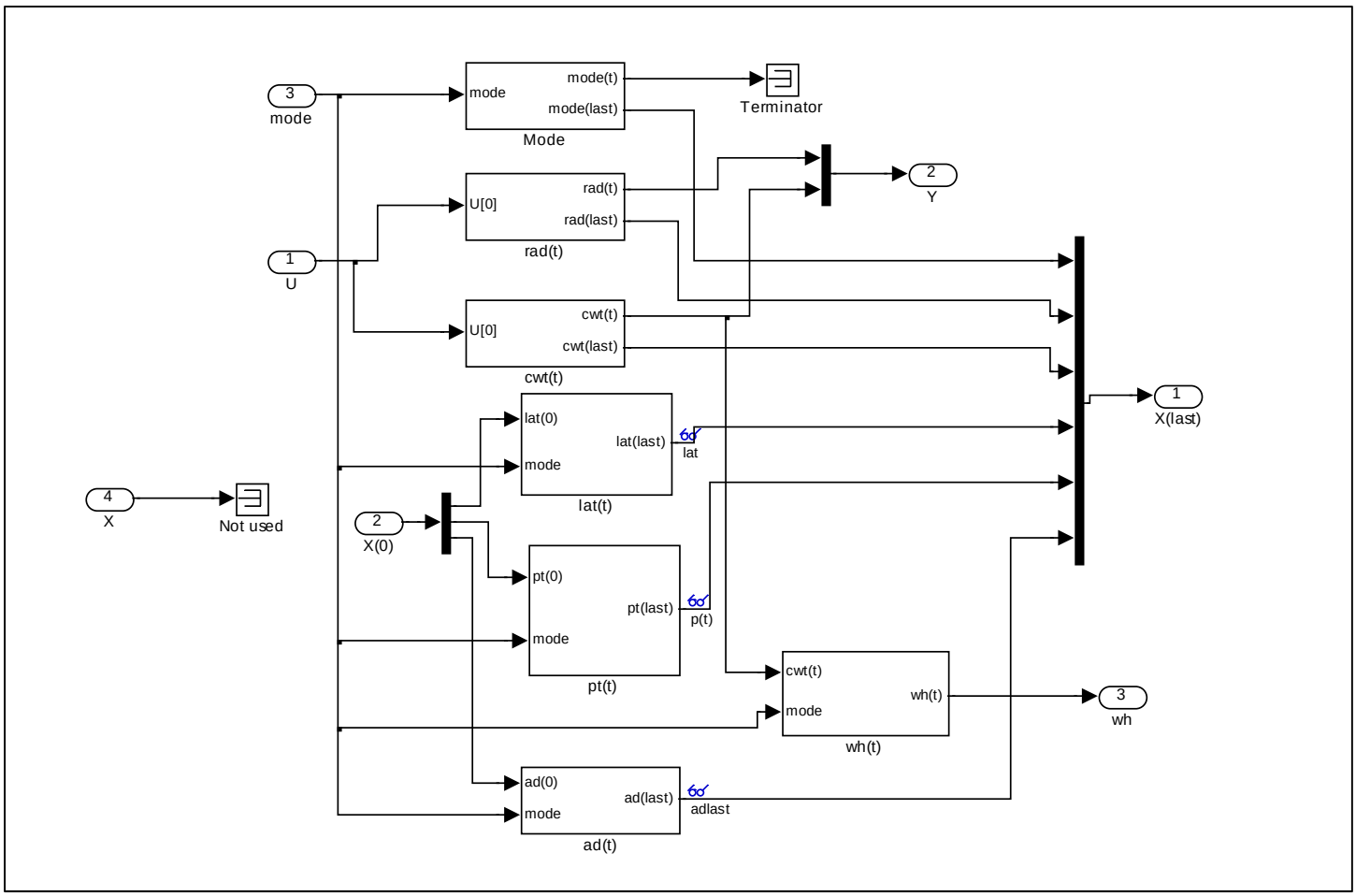

Figure 11: Trajectory Definitions overview

Figure 12 shows the Mode subsystem implementation. It simply applies a one step delay to the mode for inclusion in the X(last) vector.

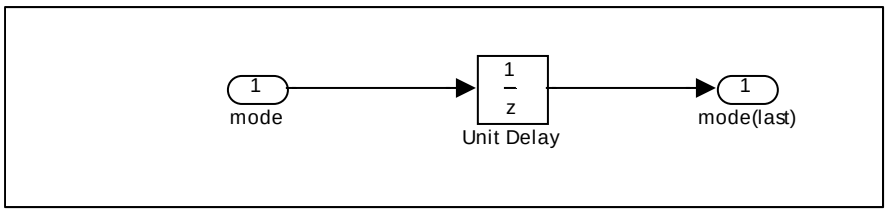

Figure 12: Trajectory Definitions/Mode implementation

The rad(t) subsystem passes through the raw continuous input. As shown in Figure 13, it has two outputs. The undelayed signal is routed to the continuous output (*Y*) vector, and the delayed version is routed to the X(last) vector.

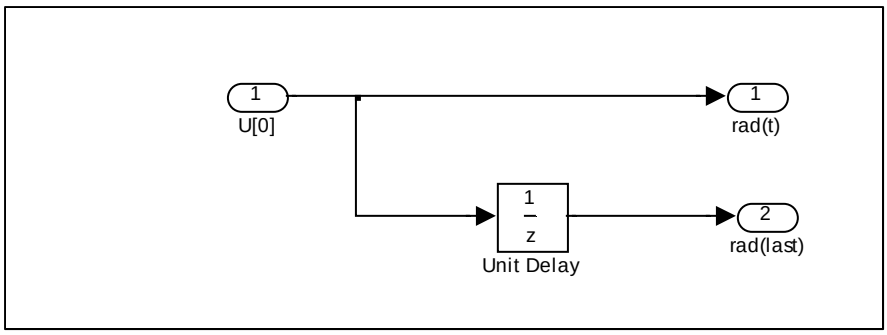

Figure 13: rad(t) trajectory implementation

The cwt(t) subsystem, shown in Figure 14, converts raw ADC values to units of weight. It also has two outputs: an undelayed signal routed to the continuous output vector and a delayed output routed to the *X*(last) vector.

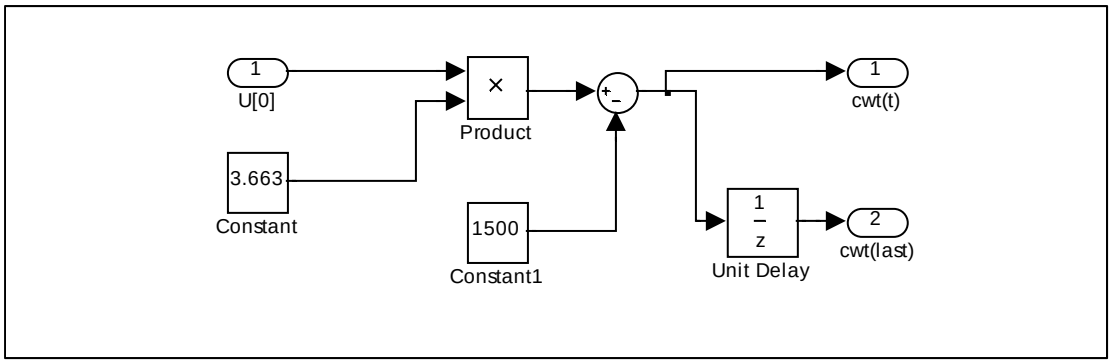

Figure 14: cwt(t) trajectory implementation

The implementations of lat(t) and pt(t) are identical. Figure 15 shows the lat(t) subsystem. The current mode is the index into a one-column lookup table that selects whether the timer is active. If the timer is active, the output is a ramp signal produced by integrating a constant from the initial condition, lat(0) or pt(0), generated by the Condition Vector subsystem. Otherwise, the output is set to zero. In either case a one time step delay is applied before the output is routed to the *X*(last) vector.

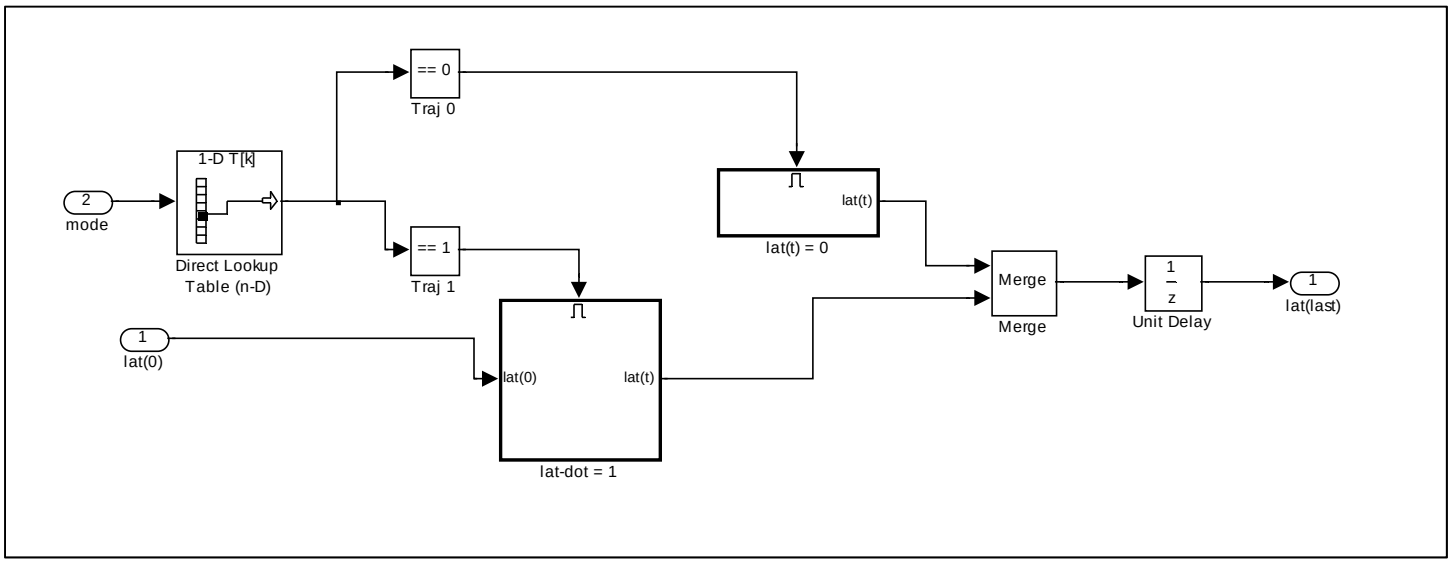

Figure 15: lat(t) trajectory implementation

The wh(t) subsystem generates and updates the vector of cwt(t) history values used in the axle calculations. Figure 16 shows the implementation of wh(t). The current mode is the index into a one-column lookup table. If the selected value is 1, the current value of cwt(t) is passed to a Delay Line block. If the selected value is 0, zero is passed to the Delay Line. The Delay Line block maintains a buffer of the last 400 values selected and outputs the buffer contents as a vector at each time step.

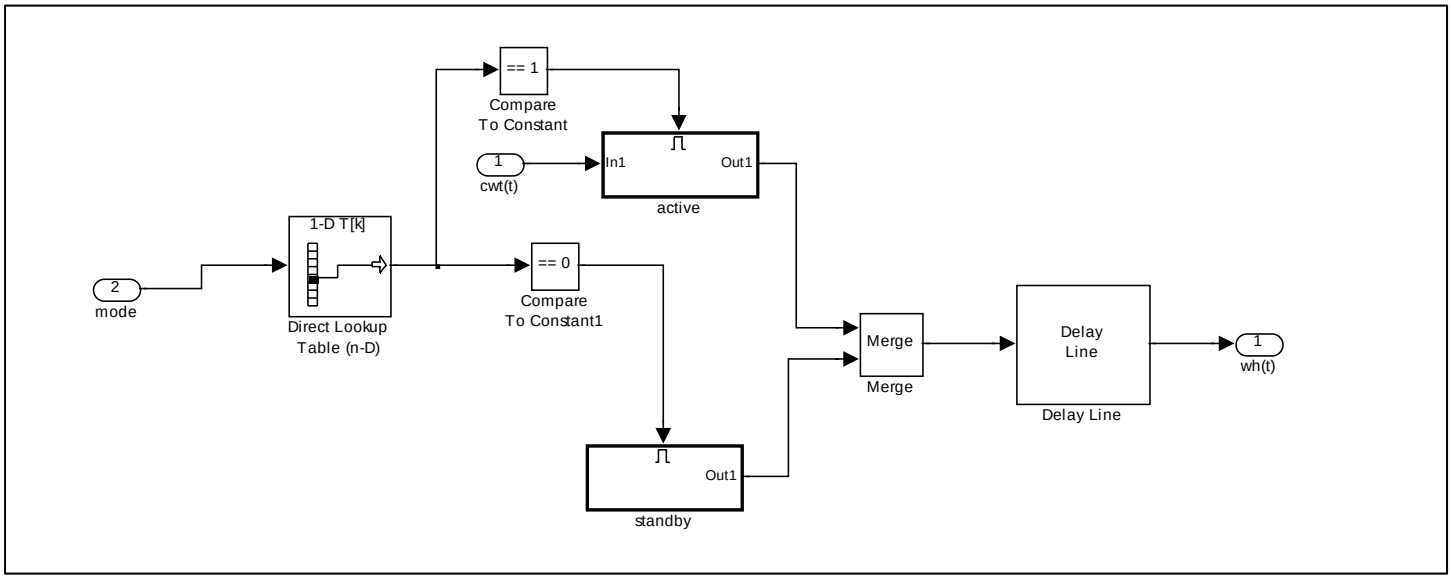

Figure 16: wh(t) trajectory implementation

The ad(t) subsystem (Figure 17) generates a signal that is either 0 or the current value of tst generated by the Condition Vector subsystem. A non-zero signal value (ad(0) = tst) indicates that an axle has been detected and that the calculations have not been performed for that axle. Otherwise the output is zero. The selected output is delayed one time step before routing to *X*(last).

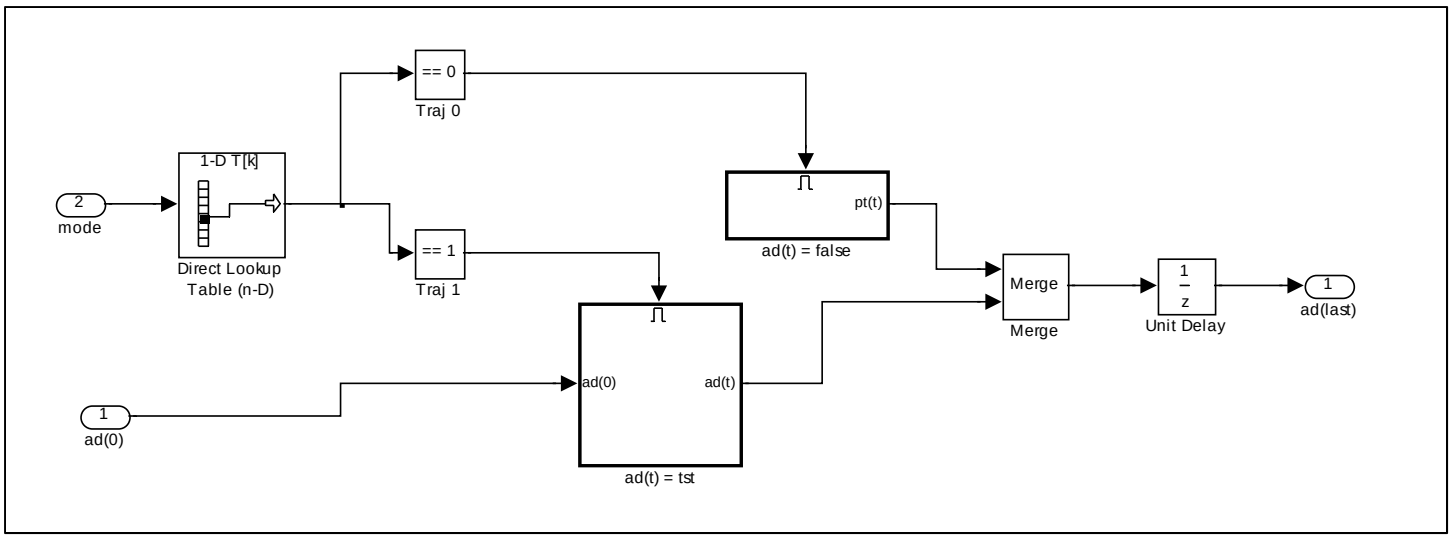

Figure 17: ad(t) trajectory implementation

Automation Considerations

The hybrid enumeration method described in [1] and [2] can be partially automated to enforce methodological rules and relieve practitioners of tedious bookkeeping tasks. However the method is inherently interactive at some level because

of the need for human interpretation of informal requirements. On the other hand, the transformation from the hybrid enumeration to an EHA can, in principle, be completely automated since the enumeration contains all the information needed to construct the EHA.

The example of a WIM DAP Simulink™ implementation demonstrates that all the information needed for manual implementation can be included in the EHA. However automation of this process requires the following:

1. Matching the level of abstraction in the declaration and refinement steps in the enumeration to the implementation framework (*i.e*., Simulink™)
2. Defining a sufficiently rich set of transformations to map all EHA elements to Simulink™ constructs.

To automate implementation in Simulink™ (and most other frameworks), the declarations (*e.g*., Tables 1 through 6) must include the following details:

1. Variable type – integer, float, enumerated, binary
2. Variable structure – scalar, array, vector
3. Overall variable range
4. Expressions defining all possible trajectories for autonomous and continuous output signals
5. Indices to be associated with discrete stimuli
6. Expressions defining values associated with discrete internal and output signals
7. Default values of discrete internal and output signals when not activated

Note that complex expressions can be specified by reference to a MatLab™ function.

Simulink™ representations can be defined for the predicates in Table 8 in terms of these basic cases:

1. Equality to any member in a set of discrete values
2. Membership in a finite interval, closed or open on either end
3. Membership in an interval, closed or open on one end and infinite on the other

For generality, it should be possible to use constant parameters in defining both declarations and predicates. Strictly speaking, the type and range of constants should be defined at specification time. Constants should be limited to parameters that would be defined at the time of deployment and not expected to change during system operation. Otherwise they need to be included as input signals.

If the considerations above are addressed thoroughly, automated implementation of the State Machine, Condition Vectors, and Trajectory Definitions subsystems is straightforward for the most part. However the following elements of the WIM DAP specification suggest the need for additional sophistication in the automation algorithm:

1. Computational dependencies among autonomous and output signals suggest the need for some built-in dependency analysis both to optimize the model implementation process and to recognize circular dependencies. This issue applies to both the Condition Vectors and Trajectory Definitions subsystems.
2. In this example (and probably in many others), the value of the clock signal used for sampling must be provided as an implicit input signal for use in computations.
3. To generate the weight history signal, a Delay Line block is used to create the required vector. In Simulink™, the frame-based output of the Delay Line cannot be multiplexed with other sample-based signals in the X-vector. Therefore the {wh(j)} signal must be routed separately throughout the model. The Delay Line idiom is likely to

occur often enough in control and monitoring systems to justify additional complexity in any automated implementation application.

Conclusion and Future Work

This report provides an example of the process for implementing a system specified via HSBS. The WIM DAP system is implemented as a Simulink™ model using information from the EHA derived in [1]. The implementation demonstrates the process of mapping the EHA data structures into a model-based programming framework. For the WIM DAP example, there are two key results:

1. The EHA produced by HSBS provides all the information needed for a complete model-based implementation.
2. The set of programming elements in Simulink™ is sufficiently rich to enable a systematic transformation from EHA tables to the implementation.

These results support the notion that, once an EHA has been produced via HSBS, implemenbtation of the specified software system can be automated, given a well defined framework comparable to Simulink™. The next step in HSBS research is development of a prototype application for automatic transformation of an EHA to a model based implementation for Simulink™ or a comparable framework.

References

1. T. Swain, *Application of Hybrid Sequence-Based Specification to a Data Acquisition Processor,* Technical Report ut-cs-11-669, University of Tennessee Department of Electrical Engineering and Computer Science, March 15, 2011.
2. J. Carter, *Sequence-Based Specification of Embedded Systems*, Ph.D. Dissertation, The University of Tennessee, December 2009.

Appendix

```
function [iaw,aaw,stdev,tsp,gsf,toc,tfl] = calcs(tst, tnd, wh, t)
%#eml
dt = 0.001;
whSize = length(wh);
tstIndex = whSize - round((t - tst)/dt);
tndIndex = whSize - round((t - tnd)/dt);
tstIndex = max(1, tstIndex);
tndIndex = min(whSize, tndIndex);
%eml.varsize('wf', 400);
wf = wh(tstIndex:tndIndex);
% compute average axle weight
wfLen = length(wf);
x0 = wf(1);
x1 = wf(150);
x2 = wf(wfLen - 1);
aaw = sum(wf) / wfLen;
% compute time on center
toc = dt * wfLen;
% compute tire speed
padLen = 2.0;
fpsToMph = 0.68;
tsp = fpsToMph * padLen / toc;
% compute integrated average weight
pt = 20;
ww = wf(pt:(wfLen - pt));
wwLen = length(ww);
iaw = sum(ww) / wwLen;
% tire footprint length
tfl = tsp * dt * wwLen;
% compute std dev
d = wf - aaw;
dd = d .* d;
rss = sum(dd(:));
stdev = sqrt(rss / wfLen);
% goodness of speed fit
dif = 0;
for i = 1:wfLen
    d0 = wf(i) - aaw;
    dif = dif + d0;
end
gsf = dif / wfLen;
```