

CONTENTS

DFB Utility	2
General Operation	3
Main Menu	4
Globals Tab	5
Stage A / B Tabs	6
Code tab.....	7
Syntax highlighting.....	8
Value converter tab	9
Log Tab.....	10
ACU and Data Ram Tab.....	11
Jump Conditions Tab	12
Hold A/B Tabs	13
Diagram Area	15
Zoom.....	15

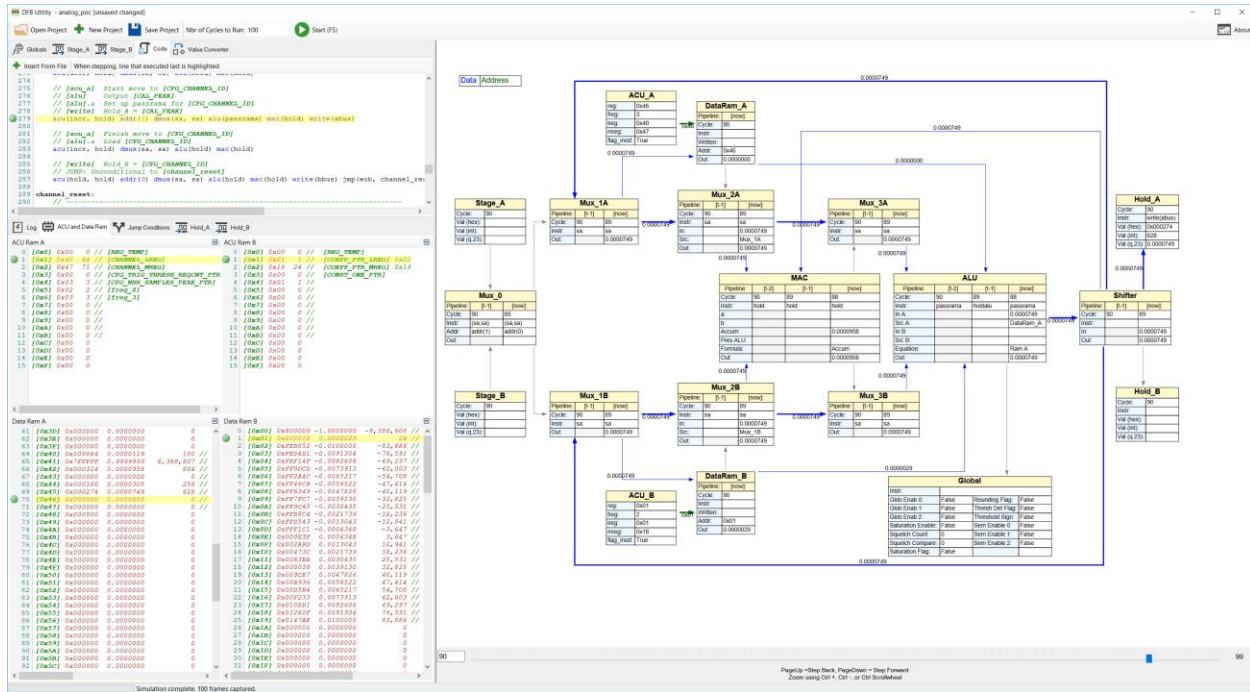
DFB UTILITY

Version 1.0

Paul Phillips

The DFB Utility is a mini development environment for developing and troubleshooting Cypress PSoC Digital Filter Block assembly code.

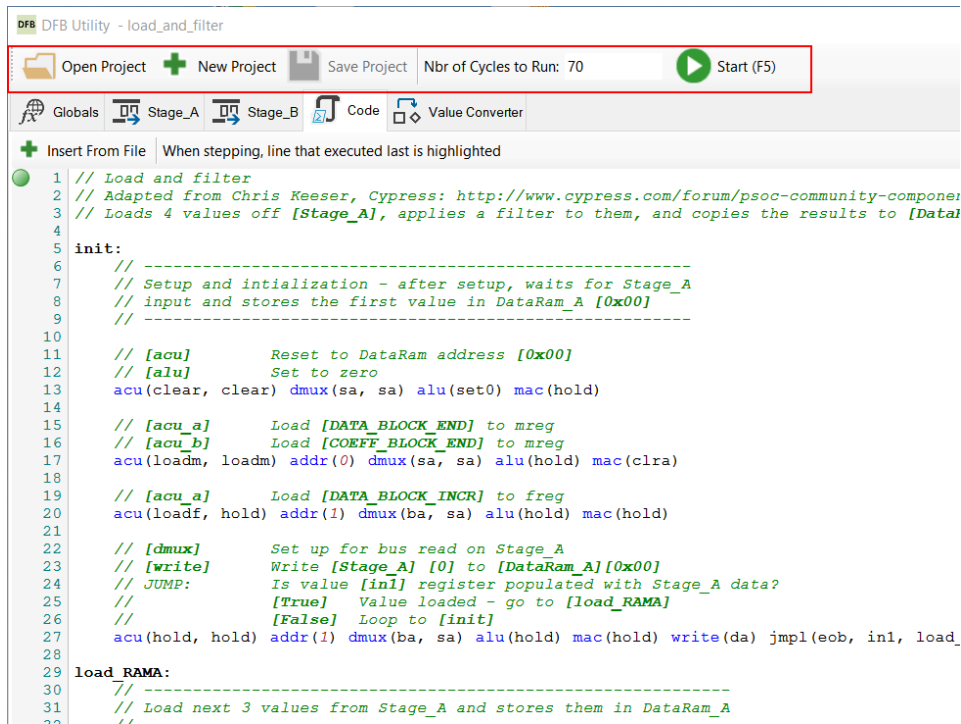
The goal of this tool is to decrease the learning curve and development time of this powerful component by providing a comprehensive view of the machine state and code at each cycle.



GENERAL OPERATION

1. Enter Stage Bus In data
2. Populate the Code tab
3. Enter the 'Nbr of Cycles to Run' value
4. Hit F5 or click the Start button
5. Once the simulation and diagram generation are complete, use Page Up and Page Down buttons to 'scrub' through the code either forward or backward
6. Make any code updates and hit F5 to re-assemble. The environment will continue from the same cycle.

MAIN MENU



1. Open Project
 - a. Click to open a *.dfbproj file.
 - b. *.dfbproj files are simple xml containing the setup information for a project in a single file:
Version, Bus1Data, Bus2Data, Code, CyclesToRun, InputSequence
2. New Project - Click to start a new project
3. Save Project – Click to save a projet
4. Nbr of Cycles to Run – enter the number of cycles to run
 - a. Note: You may hit enter in this box to start the simulation

GLOBALS TAB

DFB Utility - load_and_filter

Open Project + New Project Save Project Nbr of Cycles to Run: 70 Start (F5)

Globals Stage_A Stage_B Code Value Converter

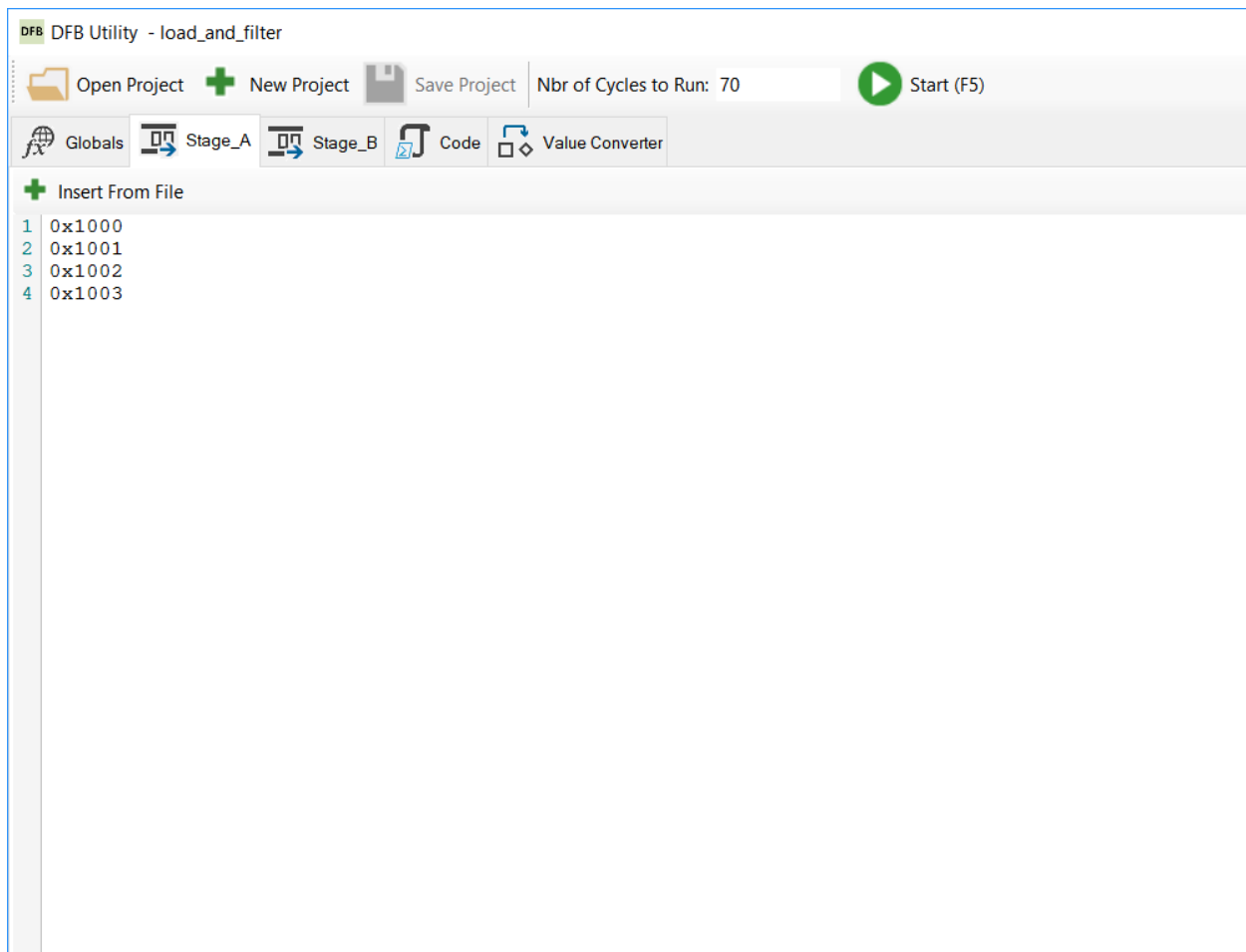
Enter a cycle number and the globals to set at that cycle

	Cycle	In 1	In 2	Sem 0	Sem 1	Sem 2
▶	0	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
*		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

The globals tab provides a way to set global values for specific cycles. These simulate inputs from the application to the DFB at specific steps

1. Cycle: The cycle number to apply the values
2. In 1 / In 2: Check to set In 1 or 2 to True at the specified cycle
3. Sem 0,1,2: Check to set semaphore 0,1, or 2 to True at the specified cycle

STAGE A / B TABS



These tabs provide simulated input data that will be read by the DFB. Enter values in hex, one line per value.

1. Insert from File: Click this to import the contents of a file into the window. The data will be saved inside the .dfbproj file.

CODE TAB

```

DFB Utility - load_and_filter

Open Project + New Project Save Project Nbr of Cycles to Run: 70 Start (F5)

Globals Stage_A Stage_B Code Value Converter

+ Insert From File When stepping, line that executed last is highlighted

5 init:
6 // -----
7 // Setup and initialization - after setup, waits for Stage_A
8 // input and stores the first value in DataRam_A [0x00]
9 // -----
10
11 // [acu]      Reset to DataRam address [0x00]
12 // [alu]      Set to zero
13 acu(clear, clear) dmux(sa, sa) alu(set0) mac(hold)
14
15 // [acu_a]    Load [DATA_BLOCK_END] to mreg
16 // [acu_b]    Load [COEFF_BLOCK_END] to mreg
17 acu(loadm, loadm) addr(0) dmux(sa, sa) alu(hold) mac(clra)
18
19 // [acu_a]    Load [DATA_BLOCK_INCR] to freg
20 acu(loadf, hold) addr(1) dmux(ba, sa) alu(hold) mac(hold)
21
22 // [dmux]     Set up for bus read on Stage_A
23 // [write]    Write [Stage_A] [0] to [DataRam_A][0x00]
24 // JUMP:      Is value [in1] register populated with Stage_A data?
25 //           [True] Value loaded - go to [load_RAM_A]
26 //           [False] Loop to [init]
27 acu(hold, hold) addr(1) dmux(ba, sa) alu(hold) mac(hold) write(da) jmpl(eob, in1, load_
28
29 load_RAM_A:
30 // -----
31 // Load next 3 values from Stage_A and stores them in DataRam_A
32 // -----
33
34 // [acu_a]    Advance the [DataRam_A] location to [0x06]: reg [0x00] + freg [0x06]
35 // [dmux]     Keep Stage_A connected to datapath
36 // [write]    Write [Stage_A] [0] to [DataRam_A][0x06]

```

This tab provides a syntax-highlighted view of the assembler source.

1. Insert from File: Click this to import the contents of a file into the window. The data will be saved inside the .dfbproj file.

As you step forward or backward in the program, the line is highlighted in yellow with a green circle bookmark:

```

+ Insert From File When stepping, line that executed last is highlighted

57 Filter:
58 // -----
59 // Apply the filter
60 // -----
61
62 // [acu_a]    Advance DataRam address by 1
63 // [acu_b]    Advance DataRam address by 1
64 // [dmux]     Keep MAC fed with DataRam values
65 // [mac].a    Load [INPUT_n] from [DataRam_A]
66 // [mac].b    Load [ALPHA] coefficient from [DataRam_B]
67 // [mac].out  No output yet
68 acu(incr, incr) dmux(sra, sra) alu(hold) mac(macc)
69
70 // [acu_a]    Advance DataRam address by 1
71 // [acu_b]    Advance DataRam address by 1
72 // [dmux]     Keep MAC fed with DataRam values
73 // [mac].a    Load [INPUT_n_MULT1] from [DataRam_A]
74 // [mac].b    Load [2*ALPHA] coefficient from [DataRam_B]
75 // [mac].out  Prior macc operation result
76 // [write]    Output calculation result to [Hold_A]
77 acu(incr, incr) addr(1) dmux(sra, sra) alu(hold) mac(macc) write(abus)
78

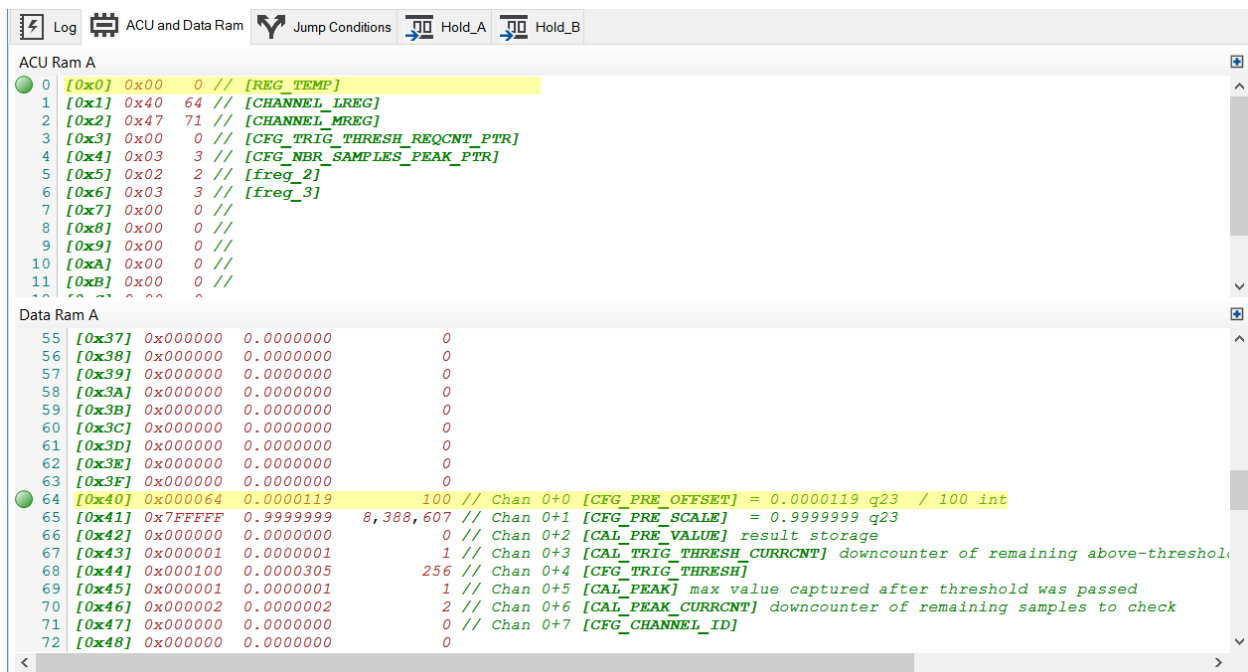
```

SYNTAX HIGHLIGHTING

1. VLIW keywords are shown in blue
2. Labels are bold
3. Comments are italic green
 - a. Text inside square brackets is bolded
4. Data values are shown in magenta

NOTE ON COMMENTS:

It is recommended to add comments next to acu, data_a, data_b entries to tag the purpose of the address, e.g. a variable name. Comments are carried through to the ACU and Data Ram tab so that you can quickly reference the current address's tag as the program executes:



```
Log ACU and Data Ram Jump Conditions Hold_A Hold_B
ACU Ram A
0 [0x0] 0x00 0 // [REG_TEMP]
1 [0x1] 0x40 64 // [CHANNEL_LREG]
2 [0x2] 0x47 71 // [CHANNEL_MREG]
3 [0x3] 0x00 0 // [CFG_TRIG_THRESH_REQCNT_PTR]
4 [0x4] 0x03 3 // [CFG_NBR_SAMPLES_PEAK_PTR]
5 [0x5] 0x02 2 // [freq_2]
6 [0x6] 0x03 3 // [freq_3]
7 [0x7] 0x00 0 //
8 [0x8] 0x00 0 //
9 [0x9] 0x00 0 //
10 [0xA] 0x00 0 //
11 [0xB] 0x00 0 //
Data Ram A
55 [0x37] 0x000000 0.000000 0
56 [0x38] 0x000000 0.000000 0
57 [0x39] 0x000000 0.000000 0
58 [0x3A] 0x000000 0.000000 0
59 [0x3B] 0x000000 0.000000 0
60 [0x3C] 0x000000 0.000000 0
61 [0x3D] 0x000000 0.000000 0
62 [0x3E] 0x000000 0.000000 0
63 [0x3F] 0x000000 0.000000 0
64 [0x40] 0x000064 0.0000119 100 // Chan 0+0 [CFG_PRE_OFFSET] = 0.0000119 q23 / 100 int
65 [0x41] 0x7FFFFFFF 0.9999999 8,388,607 // Chan 0+1 [CFG_PRE_SCALE] = 0.9999999 q23
66 [0x42] 0x000000 0.000000 0 // Chan 0+2 [CAL_PRE_VALUE] result storage
67 [0x43] 0x000001 0.000001 1 // Chan 0+3 [CAL_TRIG_THRESH_CURRCNT] downcounter of remaining above-threshold
68 [0x44] 0x000100 0.0000305 256 // Chan 0+4 [CFG_TRIG_THRESH]
69 [0x45] 0x000001 0.000001 1 // Chan 0+5 [CAL_PEAK] max value captured after threshold was passed
70 [0x46] 0x000002 0.000002 2 // Chan 0+6 [CAL_PEAK_CURRCNT] downcounter of remaining samples to check
71 [0x47] 0x000000 0.000000 0 // Chan 0+7 [CFG_CHANNEL_ID]
72 [0x48] 0x000000 0.000000 0
```

ACU comments are split by using a pipe character, to ensure that the A side and B side appear in the correct window.

VALUE CONVERTER TAB

Globals
Stage_A
Stage_B
Code
Value Converter

Enter / change values in one format to convert to the others

Hex (0x) + 6 digits	DFB Q.23	Signed Integer
0 0x001000	0 0.0004883	0 4,096
1 0x001001	1 0.0004884	1 4,097
2 0x001002	2 0.0004885	2 4,098
3 0x001003	3 0.0004886	3 4,099
	4	4

This tab lets you quickly enter a value in one of three formats and have it converted to the others. The DFB uses the following numeric ranges:

DFB Val	Signed Int	Hex Sign Exp.	Hex	Bin					
0.9999999	8,388,607	00007FFFFFFF	7FFFFFFF	0111	1111	1111	1111	1111	1111
0.0000001	1	0000000001	000001	0000	0000	0000	0000	0000	0001
0.0000000	0	0000000000	000000	0000	0000	0000	0000	0000	0000
-0.0000001	-1	FFFFFFFFFFFF	FFFFFFFF	1111	1111	1111	1111	1111	1111
-1.0000000	-8,388,608	FFFFF8000000	800000	1000	0000	0000	0000	0000	0000

Type into one of the 3 boxes to enter a value. You may also paste multiple lines into one box.

Note: If the converted values do not show, you have either entered a value that is too short or an invalid value for the DFB.

LOG TAB

```
Log ACU and Data Ram Jump Conditions Hold_A Hold_B
----- Assemble Started: 2/14/2018 1:54:30 PM -----
Control used: 59 of 64 words (92.2 %).
Data A used: 71 of 128 words (55.5 %).
Data B used: 26 of 128 words (20.3 %).
ACU used: 7 of 16 words (43.8 %).
CFSM used: 13 of 32 words (40.6 %).
----- Assemble Completed: 2/14/2018 1:54:30 PM -----
----- Simulation Started: 2/14/2018 1:54:30 PM -----
DFB Simulator version 3.0.0
----- Input data -----
Bus1 Data:
000000000000000010110000
0000000000000000100000001
00000000000000001000010001
000000000000001011000001
000000000000000010110000
0000000000000000101100001
000000000000001000010001
000000000000001011000001
00000000000000101011110
000000000000001010111100
0000000000000010000011011
000000000000010101111001
00000000000000101011110
000000000000001010111100
000000000000010000011011
000000000000010101111001
00000000000001000000111
00000000000001000000110
000000000000011000010110
00000000000010000001101
0000000000001000000111
000000000000100000011
```

This tab outputs the log from the assembler component, and also displays any errors that occur.

ACU AND DATA RAM TAB

Log	ACU and Data Ram	Jump Conditions	Hold_A	Hold_B
ACU Ram A				
0	[0x0] 0x00	0	// [REG_TEMP]	
1	[0x1] 0x40	64	// [CHANNEL_LREG]	
2	[0x2] 0x47	71	// [CHANNEL_MREG]	
3	[0x3] 0x00	0	// [CFG_TRIG_THRESH_REQCNT_PTR]	
4	[0x4] 0x03	3	// [CFG_NBR_SAMPLES_PEAK_PTR]	
5	[0x5] 0x02	2	// [freq_2]	
6	[0x6] 0x03	3	// [freq_3]	
7	[0x7] 0x00	0	//	
8	[0x8] 0x00	0	//	
9	[0x9] 0x00	0	//	
10	[0xA] 0x00	0	//	
11	[0xB] 0x00	0	//	
Data Ram A				
55	[0x37] 0x000000	0.000000	0	
56	[0x38] 0x000000	0.000000	0	
57	[0x39] 0x000000	0.000000	0	
58	[0x3A] 0x000000	0.000000	0	
59	[0x3B] 0x000000	0.000000	0	
60	[0x3C] 0x000000	0.000000	0	
61	[0x3D] 0x000000	0.000000	0	
62	[0x3E] 0x000000	0.000000	0	
63	[0x3F] 0x000000	0.000000	0	
64	[0x40] 0x000064	0.000019	100	// Chan 0+0 [CFG_PRE_OFF]
65	[0x41] 0x7FFFFFFF	0.999999	8,388,607	// Chan 0+1 [CFG_PRE_SCA]
66	[0x42] 0x000000	0.000000	0	// Chan 0+2 [CAL_PRE_VAI]
67	[0x43] 0x000001	0.000001	1	// Chan 0+3 [CAL_TRIG_TE]
68	[0x44] 0x000100	0.0000305	256	// Chan 0+4 [CFG_TRIG_TE]
69	[0x45] 0x000001	0.000001	1	// Chan 0+5 [CAL_PEAK]
70	[0x46] 0x000002	0.000002	2	// Chan 0+6 [CAL_PEAK_CU]
71	[0x47] 0x000000	0.000000	0	// Chan 0+7 [CFG_CHANNEL]
72	[0x48] 0x000000	0.000000	0	
ACU Ram B				
0	[0x0] 0x00	0	// [REG_TEMP]	
1	[0x1] 0x01	1	// [COEFF_PTR_LREG] 0x02	
2	[0x2] 0x18	24	// [COEFF_PTR_MREG] 0x19	
3	[0x3] 0x00	0	// [CONST_ONE_PTR]	
4	[0x4] 0x01	1	//	
5	[0x5] 0x00	0	//	
6	[0x6] 0x00	0	//	
7	[0x7] 0x00	0	//	
8	[0x8] 0x00	0	//	
9	[0x9] 0x00	0	//	
10	[0xA] 0x00	0	//	
11	[0xB] 0x00	0	//	
Data Ram B				
0	[0x00] 0x800000	-1.000000	-8,388,608	// [CONST_ONE]
1	[0x01] 0x000018	0.000029	24	// [NBR_SLOPE_SAMPLES]
2	[0x02] 0xFEB852	-0.010000	-83,886	// [SLOPE24_BETA_00]
3	[0x03] 0xFED4D1	-0.0091304	-76,591	// [SLOPE24_BETA_01]
4	[0x04] 0xFFE14F	-0.0082608	-69,297	// [SLOPE24_BETA_02]
5	[0x05] 0xFF0DCD	-0.0073913	-62,003	// [SLOPE24_BETA_03]
6	[0x06] 0xFF2A4C	-0.0065217	-54,708	// [SLOPE24_BETA_04]
7	[0x07] 0xFF46CA	-0.0056522	-47,414	// [SLOPE24_BETA_05]
8	[0x08] 0xFF6349	-0.0047826	-40,119	// [SLOPE24_BETA_06]
9	[0x09] 0xFF7FC7	-0.0039130	-32,825	// [SLOPE24_BETA_07]
10	[0x0A] 0xFF9C45	-0.0030435	-25,531	// [SLOPE24_BETA_08]
11	[0x0B] 0xFFB8C4	-0.0021739	-18,236	// [SLOPE24_BETA_09]
12	[0x0C] 0xFFD543	-0.0013043	-10,941	// [SLOPE24_BETA_10]
13	[0x0D] 0xFFF1C1	-0.0004348	-3,647	// [SLOPE24_BETA_11]
14	[0x0E] 0x000E3F	0.0004348	3,647	// [SLOPE24_BETA_12]
15	[0x0F] 0x002ABD	0.0013043	10,941	// [SLOPE24_BETA_13]
16	[0x10] 0x00473C	0.0021739	18,236	// [SLOPE24_BETA_14]
17	[0x11] 0x0063BB	0.0030435	25,531	// [SLOPE24_BETA_15]

This tab shows the ACU and DataRam values after the current cycle executed. The current address of each is highlighted in yellow with a green circle bookmark.

Either A or B sides may be collapsed.

Comments are carried over from the Code as entered.

JUMP CONDITIONS TAB

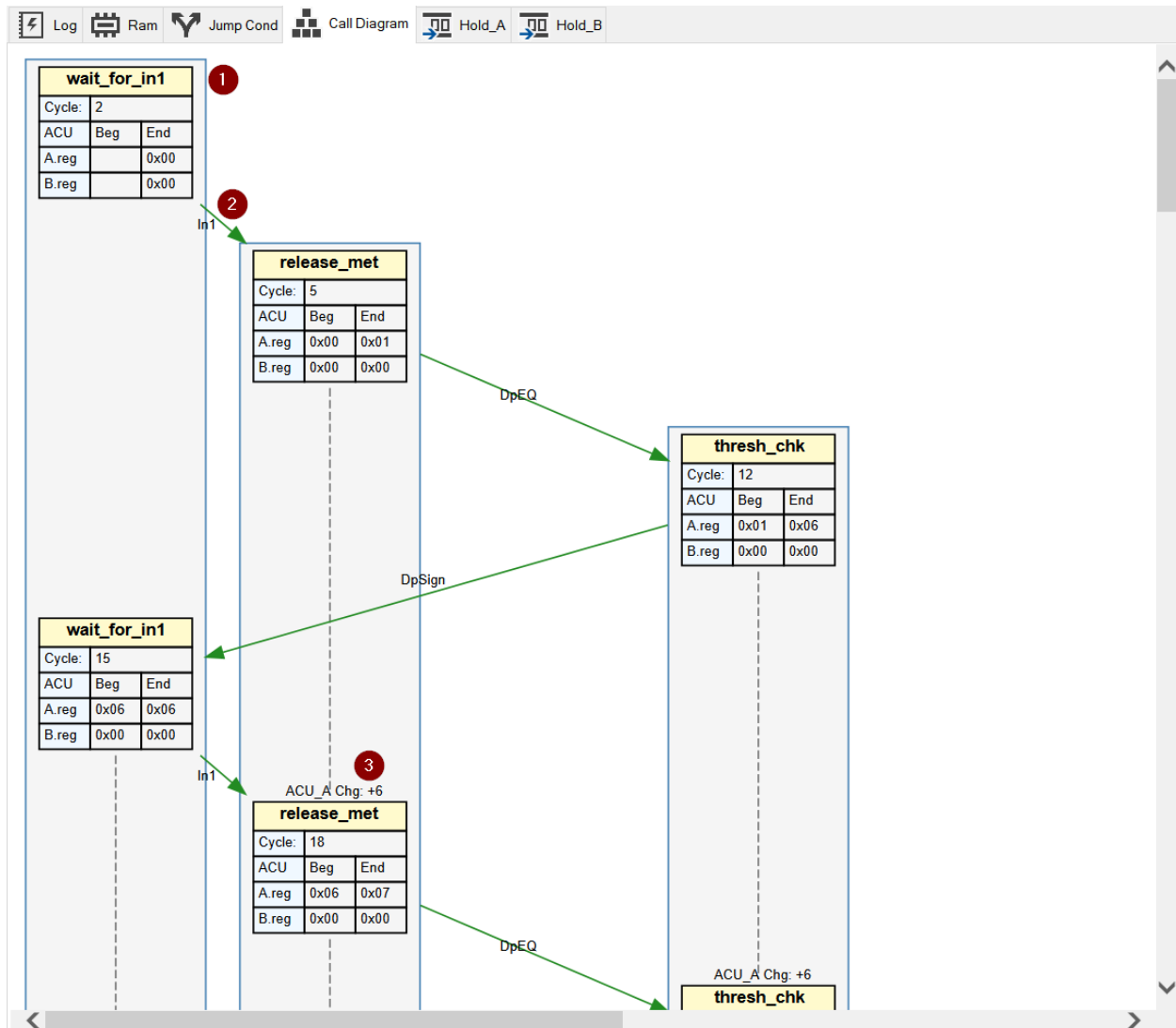
Log	ACU and Data Ram	Jump Conditions	Hold_A	Hold_B
1	Jump Cond	[now]		
2		-----		
3		eob True		
4				
5		dpsign		
6				
7		dpthresh		
8				
9		dpeq		
10				
11		acuaeq True		
12		acubeq True		
13				
14		in1		
15		in2		
16				
17		sem_0		
18		sem_en0		
19				
20		sem_1		
21		sem_en1		
22				
23		sem_2		
24		sem_en2		
25				
26		glob_in1		
27		glob_en1		
28				
29		glob_in2		
30		glob_en2		
31				
32		sat		
33		sat_en		
34				
35				

This tab shows the current value of the conditions which may be used as a true branch in jump instructions.

For example, in the below you see that dpeq = True in the jump conditions tab when the code reaches the highlighted line. Dpeq (datapath output equals zero) is the jump condition for true branch, so the code will jump to peak_calc_met instead of falling through to thresh_chk:

Globals	Stage_A	Stage_B	Code	Value Converter
+ Insert From File When stepping, line that executed last is highlighted				
97			<code>// [alu] Pipeline wait</code>	
98			<code>acu(hold, hold) dmux(sa, sa) alu(hold) mac(hold)</code>	
99				
100			<code>// [CAL_TRIG_THRESH_CURRCNT] == [zero]?</code>	
101			<code>// [True] Threshold count has been met - go to [slope_calc]</code>	
102			<code>// [False] Threshold count has not been met - go to [thresh_chk]</code>	
103			<code>acu(hold, hold) dmux(sa, sa) alu(hold) mac(hold) jmp(eob, dpeq, peak_calc_met)</code>	
104				
105			<code>thresh_chk:</code>	
106			<code>//-----</code>	
107			<code>// Check the scaled signal against the threshold</code>	
<				
Log	ACU and Data Ram	Jump Conditions	Hold_A	Hold_B
1	Jump Cond	[now]		
2		-----		
3		eob True		
4				
5		dpsign		
6				
7		dpthresh True		
8				
9		dpeq True		
10				
11		acuaeq		
12		acubeq True		
13				
14		in1		
15		in2		
16				
17		sem_0		
18		sem_en0		
19				
20				

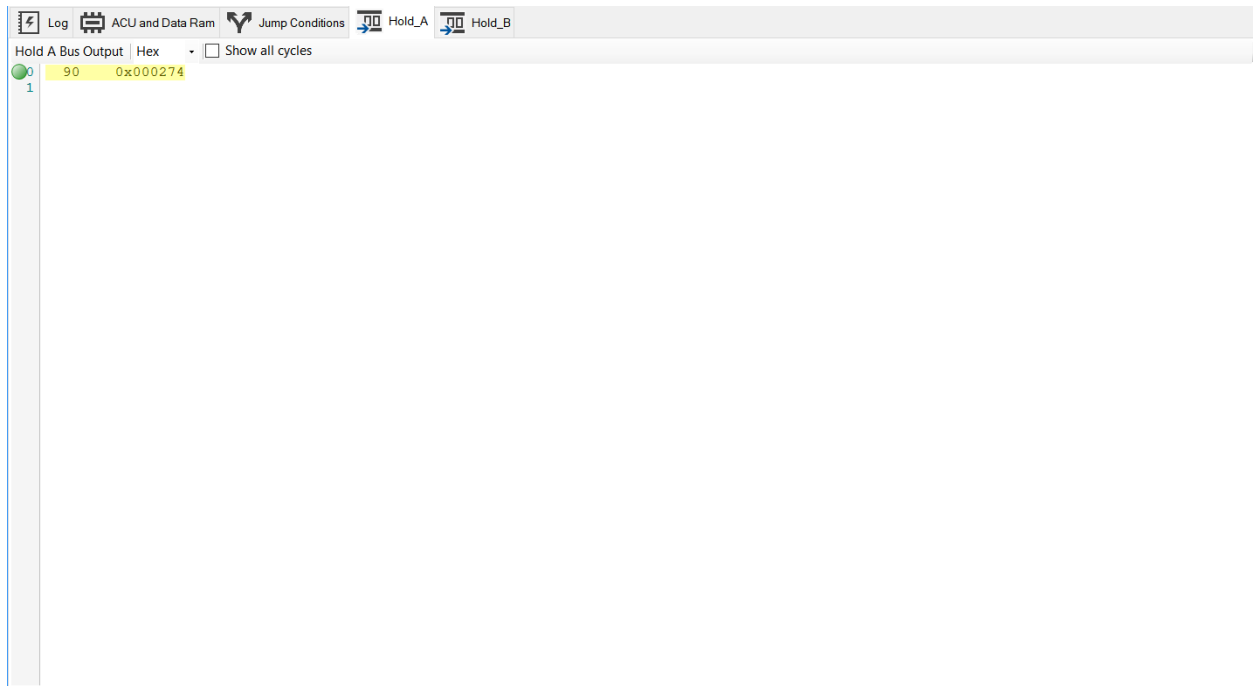
CALL DIAGRAM TAB



The call diagram will show the DFB program's flow.

- Each label appears in its own vertical lane
 - ACU address is shown as of the entrance to the label (Beg) and at the label exit (jump instruction - End)
- Each branch 'true' condition is listed on the line connecting the calls
- The change in ACU address from one call to the next is displayed
 - This helps the developer to ensure that the DataRam has been positioned correctly on each label call entry
 - For example, if the DataRam is organized for multi-channel processing where each channel has 6 coefficients and values, we expect to start a given label at either the start position (e.g. 0x00) , or start position + [6 * current channel number] (e.g. 0x00, 0x06, 0x0C, 0x12, etc)

HOLD A/B TABS

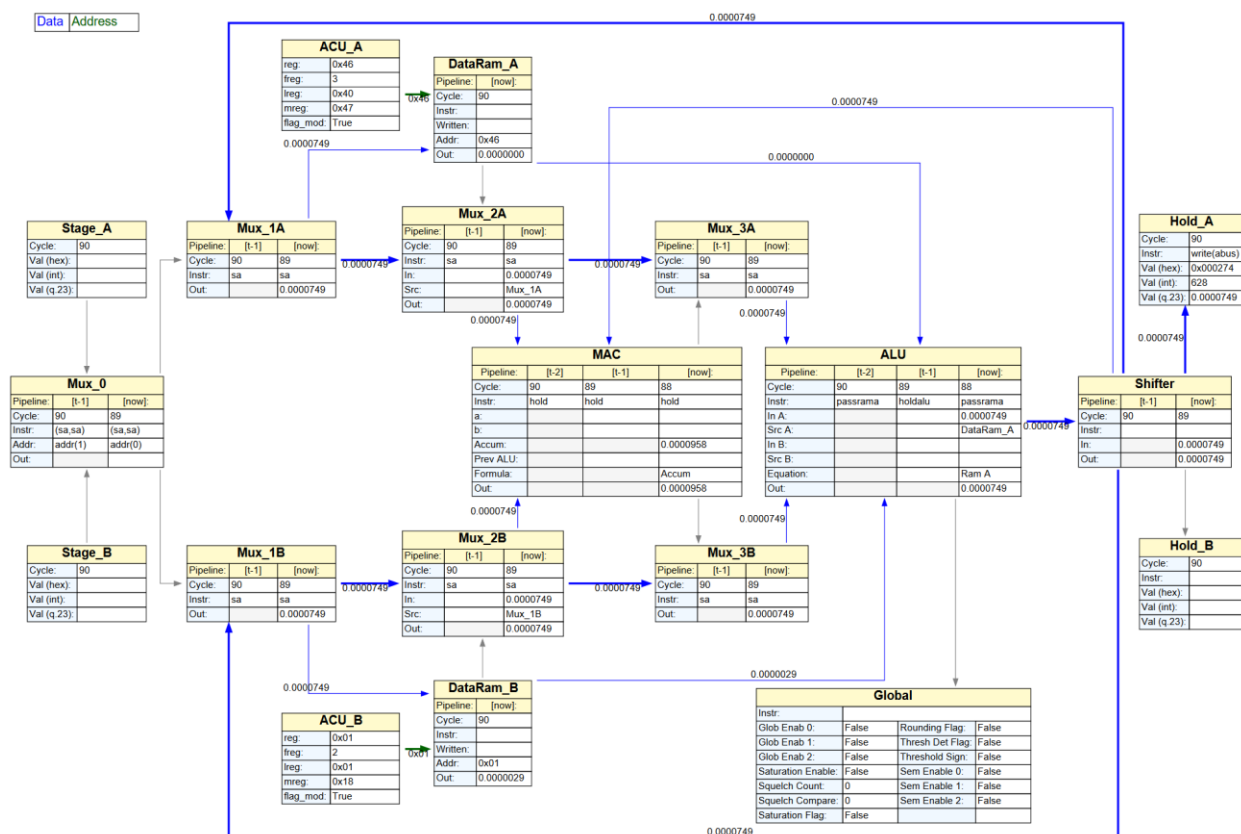


These tabs show the bus out values put into Hold_A and Hold_B. You may choose one of the 3 value formats.

1. Show all Cycles:
 - a. Checked: shows all cycles regardless of whether there is an output value
 - b. Unchecked: only shows cycles with output values

The current cycle is highlighted in yellow with a green circle bookmark.

DIAGRAM AREA



This tab displays a state diagram for the current cycle.

- Gray lines indicate physical datapath connections that are not active for the given mux instruction
- Light blue lines indicate physical datapath connections that are active for the given mux instruction but not being used
- Bold blue lines indicate physical datapath connections that are active for the given mux instruction and are being used by one of the devices
- Green lines indicate address buses

Values are displayed in DFB q.23 format for all internal signals. Hex and unsigned integer are shown on Stage and Hold in/out buses.

ZOOM

Use Ctrl + or Ctrl - to zoom in and out of the diagram.

Use Ctrl-0 to reset to default

Changing the window size will zoom the diagram to fit