

#### Changing the Embedded World<sup>™</sup>

### Module 3: Getting Started Debugging

PERSONAL

ACCESS

ENTERPRISE

METRO

CORE



#### **Module Objectives:**

#### **Section 1: Introduce Debugging Techniques**

Section 2: PSoC In-Circuit Emulator (ICE)

Section 3: Hands on Debugging a System with PSoC MCU

- Breakpoints
- Watch Variables
- Dynamic Event Points



Let's Run our Program

## In Circuit Emulators (ICE) required for "real application" testing

Most programs don't work the first time...

You need to test the program in circuit

Debugging allows the developer to monitor the application code line by line as it is running on the circuit.



Off by one –

**Memory corruption** 

Timing interrupts – Where did the code go?

**Stray pointers** 

Hardware design errors

**Peripheral errors** 



### **PSoC Debugging Features Overview**

Both C and Assembly level debugging Real time (24MHz) In Circuit Emulation for all part types

- Trace Buffer 128K Deep
- **Breakpoints**
- Watch Variables
- Viewing Registers, RAM, Flash, CPU registers

**Dynamic Event Points – more than a Breakpoint** 

- Break
- Trace On
- Trace Off
- External trigger



### Debugging Features – Code Debugging

#### Assembly and C level debugging

#### Assembly –

- Singular step program execution capability
- Displays Instruction in trace buffer
- Full breakpoint capability in assembly

#### Source level C -

- Single step instruction
- Step over a procedure
- Step out of a procedure
- C level breakpoint capability



### Debugging Features - Trace Buffer

### Trace Buffer 128K bytes – (overwrites) Default status is Trace buffer "ON" Selectable fields – (Debug→Trace mode)

- PC Program Counter
- Time Stamps
- SP Stack Pointer
- A Accumulator
- External pins on ICE

#### **On/Off capability with Dynamic Event Points**



### Debugging Features -Breakpoints

Halts program execution, provides the user with real time control of program execution.

Selectable via application editor or debugger

Selectable by clicking in left margin of code

**Breakpoint manager window** 



### Debugging Features -Watch Variables

# Provides the user the ability to View Global variables

#### Watch Variables may be modified

Watch Variables automatically recalled





#### **Debugging Features - Viewable Items**

#### Access to I/O Registers, RAM, Flash, CPU registers

CPU Registers – PC,SP,X, A, Flags

#### Bank 0 and Bank 1 viewable and modifiable

- 512 configuration registers may be changed
- Debugging capability to reconfigure your device while debugging
- Bank 0 Port \_2\_Data output to LCD's

#### **RAM - Modifiable and viewable when program halts**

#### Flash – Viewable Not modifiable



### Debugging Features -Dynamic Event Points

#### Advanced emulation capabilities –

- Features comparable to \$1,000+ emulators
   Provides limitless conditional testing
- Complete conditional TRACE capture
- External triggering
- Provides the capability to sequence complex test scenarios

#### Similar to a hardware logic analyzer



# Why Use Dynamic Event Points?

ENTERPRISE

METRO

CORE

ACCESS

PERSONAL

## Problem – How to monitor a complex sequence of events?

- A standard emulator provides the capability to look at one condition and perform one event a Break.
- Dynamic Event Points Can help you answer "What happens the 20<sup>th</sup> time through the loop when the Variable is equal to 99h?"
- We will perform this level of testing in the hands on section
- Dynamic Event Points Debugging a sequence of events, instruction by instruction
- Each event is a mini if/then if a condition occurs, then take one or more actions + go to the next conditions



**Dynamic Event Points** 

#### **Typical Dynamic Event Point Usages**

- Find a stack overflow SP
- Trace a specific range of code PC
- Find when a memory location is corrupted
- Find when an IO address is written
- Drive an external signal in interrupt
- Measure interrupt latency



### **Dynamic Event Points**

#### **Typical Dynamic Event Point Usages (cont.)**

- Break the Nth time a line of code is executed
- Detect jump or call out of program image
- Look for IOX access with specific address
- Break on carry flag status
- Break on signals from customer target board
- Wait for certain number of instructions
- Count sleep periods



### Dynamic Event Points: Main window

Deb	ugger Events		
	8 Bit Thread	👕 State Logic	16 Bit Thread
0	Disabled	Disabled	Disabled
1	Disabled	Disabled	Disabled
2	Disabled	Disabled	Disabled
3	Disabled	Disabled	Disabled
	Enable 8 bit Thread	State Logic Next state: Break 0 Trace On	Enable 16 bit Thread
	Input Select:     High Compare       Input Mask:     Iff	Match count: ☐ Trace Off 1 ☐ External trigger	0000     <=     PC16     <=     0000
	BITFIELD Help: Bits 0-7  Bit-0 (0x01) RAM read flag (active low) cmp Hi= cmp Lo= 00 Bit-1 (0x02) RAM write flag (active low) cmp Hi= cmp Lo= 00		uP Program Counter (PC16)
	Help	OK Cancel	Apply Clear All

PERSONAL () ACCESS ()

ENTERPRISE

CORE

METRO



**Conditions – IF ...** 

#### The test case initialization includes the capability to specify many different inputs and then add conditional logic to them!

#### The IF – possible inputs

- Accumulator, Stack Pointer, X, PC
- External logic pins
- Data address (Memory or I/O)
- Data value (Memory or I/O)
- Bit Field Global Int, Extended I/O, R/W, Carry, Zero Flags
- Count

**Conditions:** Greater Than / Less Than / Equal To





### Actions - ... THEN

#### **Possible Actions:**

#### Test sequence (event)complete control:

- ALWAYS go to next state
- Skip a state
- Sequence on the same event

#### **OPTIONS for each event:**

- Turn trace off
- Turn trace on
- Blip external trigger pin
- Break

#### CYPRESS Dynamic Event Points: Connecting From Lost Mile To First Ail Chnical Nuts and Bolts

#### Multiple threads – 8/16 bit

#### 16 bit threads

- **•All CPU registers, including PC**
- Can monitor address/data at same time
- No masking of bit fields

#### 8 bit threads

- Contains bit field IOX bit, global int, ram/mem read/write
- Can mask off unwanted bits
- No PC support
- Threads can be logically combined

Test suites can be ordered with complete flexibility



### Execute Project Within Debugger

The last section of this Module will be hands on. We will use the debugger to find the system bug and demonstrate the ICE features

Steps: Switch to the Debugger – What's Different?



Connect to the ICE

**→**|+

Download the project

Let's work!



### Debugging – Download actions

 $\Rightarrow$  Download the GettingStartedproject .rom file to the Pod by clicking the Download to Emulator icon .

The system automatically downloads your project .rom file located in the ...\output folder of your project directory. A progress indicator will report download status.

Upon successful connection, you will receive notification and a green light displaying a status of Connected will display in the lower-right corner of the subsystem.



### **Emulator Download**

#### Speed has been greatly improved for

- Emulator download
- Device programming

Download to Emulator	Download to Emulator					
Verifying						
67% complete						
Cancel						





### Explore the Debugging Window:

- ⇒The debugging window has many useful features: Register Memory space, Watch Variable list, output files, source files, and CPU registers (see the device Data Sheet, section 2.0 for details).
- In the status bar (at the bottom of the screen)you will find ICE connection indication, debugger target state information, and (in blue) Accumulator, X register, Stack Pointer, Program Counter, and Flag register values, as well as the line number of the current code.

💥 👐



### Execute Project Within Debugger



#### $\Rightarrow$ Explore the Debugging Window:

Bit www Pork Confo Buld Debug Program Tool Wrydw Hele         Bit www Pork Confo Buld Debug Program Tool Wrydw Hele         File Charles Conformation Confor	🕂 Example_DAC_output_28	pin - PSoC Designer	_	B ×
Image: State in the state	File Edit View Project Cor	rfig Build Debug Program Tools Window Help		
Image: Dec. Dec. Dec. Dec. Dec. Dec. Dec. Dec.	🛛 🔠 🗅 🛥 🖬 🕼 🕺 🕺 🖻	😂 伦 🕆 🖂 🕂 🖿 🗰 🞑 💓 🖅 🔎 🗰 🦀 🥵 🎌		
Image: Doct control from the band law         Image: Doct cont band law         Image: Doct	存存   三 🎽 🔺 🎘 🔩 🆽	#4 \$\$ \$\$ \$\$ \$\$ \$\$ \$\$ \$\$ \$\$ \$\$ \$\$ \$\$ \$\$ \$\$		
Formple Jong (and the second of the form of the fo	AX			
i Boldrein       i Condensition       i Condens	Example_DAC_outp		Current	
Image: State of the state	🖨 🔄 Source Files			
Image: Start in the start		Interrupt 10:		
<pre>i Headers i Headers i</pre>	main.asm	// call void_handler	PLAGS 00	
image:	Headers	reti	SP 00	
0 mininger Production       implementation = Read Colv         1 mininger Production       implementation = Read Colv         1 mininger Production       implementation = Read Colv         1 mininger Production       implementation         1	Elibrary Source		× 40	
Intervent       Intervent         Intervent		ret Main.asm - Read Only		
Image in the property of the pr		interru ; **********************************		
image: provide in the image: provide in the provide into the provide into the provide into the device into analog or the provide into the device into the provide into the provide into the provide into the device into the provide into the device into the provide into the provide into the device into the provide into the device into the provide into the device into the provide provide into the provide into the provide i		ret ;		
Image: Start in the start		Interru ;		
Interrup       // metric		7/ ; Purpose Fet : To demonstrate the operation of the DAC user module of the P		
Image: Start Star		Interru ;		
Internal <pre>             / Function             / Internal             / Interna</pre>			, 	
<pre>:Upon program execution all hardware settings from the device is loaded into the device and main.exe is is executed. The swalls is loaded into the device and main.executed. The swalls is loaded into the device and main.executed. The swalls is the AC outputs on cold of possible analog voites other is the AC outputs on cold of possible analog voites and go your dege on the the displays the value on the IB display. The value is in signs if or the times. The times modules intervunt is used to update if (0 to 60) to feed the DAC module allowing easy voitege obset imultimeter.</pre>		Interru ; Function		
<pre>pladed into the device and interes de sected. The example</pre>		// ;Upon program execution all hardware settings from the device	<u>C</u> urrent: 00	
		; loaded into the device and main.asm is executed. The example		CC 🔺
Image: Start       Image: Start <td< td=""><td></td><td></td><td>08 CC CC CC CC CC CC</td><td>cc</td></td<>			08 CC CC CC CC CC CC	cc
istract with the analog output centers' around analog pround joynem clock the divide by 16 (24%) and 16 (24%) to prod i for the timer. The timer modules interupt is used to update i (0 to 6) to feed the DAC module allowing easy voltage obser juilineter. ; (1 col) to feed the DAC module allowing easy voltage obser juilineter. ; (1 col) to feed the DAC module allowing easy voltage obser juilineter. ; (1 col) to feed the DAC module allowing easy voltage obser juilineter. ; (1 round Connections ; (1 is or compatible hardware. For proper operation a multime i rest Output       18 C C C C C C C C C C C C C C C C C C C		Or ; The DAC outputs one of 64 possible analog voltages onto the		cc
is gystem clock is divided by 16 (24V_1) and 16 (24V_2) to prote is gystem clock is divides interrupt is used to update (0 to 53) to feed the DAC module allowing easy voltage obset imilitanter.       20 CC		and ; format with the analog output centered around analog ground	18 CC CC CC CC CC CC CC	CC
import in the timer. The timer modules interrupt is used to update         import in the timer. The timer module allowing easy voltage obser         import in the timer.         import in the timer. <td< td=""><td></td><td>• or ;system clock is divided by 16 (24V_1) and 16 (24V_2) to prod</td><td>20 CC CC CC CC CC CC CC</td><td>CC</td></td<>		• or ;system clock is divided by 16 (24V_1) and 16 (24V_2) to prod	20 CC CC CC CC CC CC CC	CC
imilianter.       imilianter.		* mov ; for the timer. The timer modules interupt is used to update	28 CC CC CC CC CC CC CC	CC
i       i		; multimeter.	30 CC CC CC CC CC CC CC	CC
image: construction of the second of the				
i Circuit Connections   i Circuit Connectio				
i i i i i i i i i i i i i i i i i i i		; Circuit Connections		cc II
Image: Provide of comparison to introduce to proper operation of and other and the second of		; This application note runs on the PSoC pup board supplied wi	58 CC CC CC CC CC CC	CC
Files Output     RAM D Regs     Ram D Regs <t< td=""><td></td><td></td><td></td><td>cc-ll</td></t<>				cc-ll
Image: Product     Image: Prod				
Image: Start     Image: Star	Files Output		RAM IO Regs Hash	
Image: Start     Image: Star	×			
Build Debug     Image: Start				
For Help, press F1			Ολάφαει	·
For Help, press F1				
For Help, press F1				
Image: Second and Seco				
For Help, press F1       Ln 23       A: 00       X: 40       SP: 00       Halted       Idle Connected       NUM         Image: Start       Image: Start </td <td></td> <td></td> <td></td> <td></td>				
For Help, press F1       Ln 23       A: 00       X: 40       SP: 00       PE: 00       Halted       Icle       Connected       NUM         Image: Start       Image	Build Debug			
For Help, press F1 Ln 23 At 00 X: 40 SP: 00 PC: 0039 F: 00 Heltsd Idle Connected NUM	Band Debug			
	For Help, press F1	Ln 23 A: 00 X: 40 SP: 00 PC: 003	9 F: 00 Halted Idle Connected NUM	4   I
DEDSONAL ACCESS ENTERDEDISE METRO CODE	🏽 🔚 Start 🔢 🙆 😂 🏷 🖄	DE Tutorial Kimber 📑 Example_DAC_o		23 PM
			$\vee \vee \vee$	
		( PERSON	AL ACCESS A ENTERPRISE A METRO	0 000



### **Debugging Overview**

#### Following is a summary of our debug strategy:

- First, Compile and Link the project to ensure no run time errors.
- Second, Execute the program (RUN) with the PSoC pup connected.
- Verify output (on LED's) to expected output.

System operation should be the full range of 8 bit DAC output at a  $\frac{1}{4}$  second update rate showing on the LED's. Starting at the high end and proceeding down to the digital input of zero.



### Debugging Features – Code Debugging

#### Process:

Set breakpoints on both routines to see the decrementing of the OutputV value as well as the transfer to the Accumulator.

The Watch Variable from the code that we will view/monitor is the "OutputV" value. This value should cycle the entire 8 bit DAC range and then reset again.

We will see the output to Port 2 register.

Lastly, We will introduce Dynamic Event Points





### Debugging Features -Breakpoints

#### **Breakpoints**

This feature of PSoC Designer allows you to stop program execution at predetermined address locations. When a break is encountered, the program is halted at the address of the break, without executing the address's code. Once halted, the program can be restarted using any of the available menu/icon options.

For our example we will set two breakpoints:

Open the *main.asm* file by highlighting it in the source tree. (If the source tree is not showing, access <u>V</u>iew >> P<u>r</u>oject.)

Scroll down in the file to the statement: M8C\_EnableGInt.

⇒ Go to the left margin next to this statement and left click. A red dot will signify that you have just set a breakpoint.

CORE



### Debugging Features -Breakpoints

#### **Breakpoints**

second breakpoint we will set in the RCTINT.asm file.  $\Rightarrow$  Scroll down to the statement "dec [outputV]"  $\Rightarrow$  Click on the left margin next to this statement. A red dot will signify that you have set a breakpoint.

#### **Breakpoint Manager**

To view all the breakpoints you have set, access <u>Debug >> Breakpoints</u>.

 $\Rightarrow$  Press OK to close the Breakpoint Dialog Box.

Breakpoints		
Active breakpoints:		
main.asm 21 rtcint.asm 48		
inclinicasiin 40		
I		
Denson 1	Demous All	OK





### Debugging Features -Watch Variables

The ICE provides the ability to select variables of interest (from the users program) that can be monitored real

(from the users program) that can be monitored real time.

We will set a watch on the variable "OutputV." The address for this variable is 0Eh. This can be found in the Trace window or in the *output.mp* file.

 $\Rightarrow$  To set OutputV as a Watch Variable, access <u>Debug >> Watch Variables</u> and fill in the following details:





### **Trace Window**

#### 3 Display Options Trace Display Can Be:

- Saved to a file
- Viewed, saved and printed as HTML report



PSoC Designer Trace Report

C:\Program Files\Cypress MicroSystems\PSoC Designer\Examples\Example\_Dynamic\_PWM\trace wed Apr 17 17:22:01 2002

	Trace Report							
Seq	PC 	INSTRUCTION	 DATA 	A 	x 	 SP 	 FLAGS- -	 EXTERNAL 
0	01C8	OR REG[3Bh], 01h	FF	01	00	06	00	00000000
1	0106	CALL_PWM16_4_Start	08	01	00	06	00	00000000
2	0269	RET	FF	01	00	04	00	00000000
3	0266	OR REG[33h], 01h	FF	01	00	06	00	00000000
4	0104	CALL PWM16_3_Start	06	01	00	06	00	00000000
5	0307	RET	FF	01	00	04	00	00000000
6	0304	OR REG[2Bh], 01h	FF	01	00	06	00	00000000
7	0102	CALL_PWM16_2_Start	04	01	00	06	00	00000000
8	03A5	RET	FF	01	00	04	00	00000000
9	03A2	OR REG[23h], 01h	FF	01	00	06	00	00000000

PC	INSTRUCTION	DATA	A	×	SP	FLAGS	EXTERNAL
012A	OR REG[23h], 01h	FF	07	11	06	00	00000000
010D	CALL_PWM16_5_Start	OF	07	11	06	00	00000000
06A9	RET	FF	07	11	04	00	00000000
06A8	POPA	07	07	11	06	00	00000000
06A7	POPX	Save			דק	00	00000000
06A4	OR M[01h], 02h	Save Ar			98	00	00000000
06A2	AND F, EFh	Jave As		<b>-</b> -	98	00	00000000
069F	OR REG[E1h], FFh	view Trace Report			_08	10	00000000
069C	AND REG[E1h], 00h	00	C2	C2	08	12	00000000
069A	0B E 10b	00	£2	C2	08	12	0000000



### Watch Window

Global Variable addresses are automatically entered/updated each time the project is downloaded to the ICE

Global Variables can be selected from the << Globals selection list

> Name, Address, Type and Memory Area are automatically set

#### Check box to select Watch Variable



PERSONAL

ACCESS

**ENTERPRISE** 

METRO

CORE



### ICE Status Bar Read Left to Right

PERSONAL

ACCESS

ENTERPRISE

Accumulator Index Stack Pointer **Program Counter** Flag **Actual CPU frequency Program execution status ICE** communication status ICE connection status Carry and Zero flags

A: 01 X: 00 SP: 04 PC: 0108 F: 00 3.01 MHz Halted Idle Connected

C: 0, Z: 0

CORE

METRO



### **C** Compiler Project Settings

### Macro Defines Enable Multiply/Accumulate (MAC) Optimize Math Functions for Speed

roject Se	ttings			×
Compiler	Device	Editor 🛛 Linker	r Programmer	
Macro d	efines:			
Macro u	ndefines			
💿 Enal	ble Image	Craft		
🔽 Ena	ble MAC	🔽 Optimize	math functions fo	r speed
🔲 Ena	ble Code I	Compression		
				Cancel
				Cancer

METRO



### **C** Interrupts

#### **C** Interrupts are Supported

#### #pragma interrupt\_handler <name> \*

- reti is used instead of ret to return from the function
- Virtual registers used by the function are automatically saved and restored
- If another function is called from the interrupt handler all virtual registers are saved and restored

PERSONAL

ACCESS

ENTERPRISE

METRO

CORE

#### Additional Information Available in the C Language Compiler User Guide



#### **Debugging Features - Execute the Program**

We are now ready to start execution of the program. Our example should stop on the first breakpoint in *main.asm*. A yellow arrow will point to the M8C\_EnableGInt line of code when this happens.

Use the "Step Into..." functions to execute the next several instructions. Watch what happens to outputV, the Accumulator and the LEDs on the PuP at each step.
Click the Green Run icon to execute the program again. The program will stop on the second breakpoint in RTC*INT.asm*, "dec [outputV]"



🔿 📲 🚹 🔁 🗗 🚺 🔍 🖱



🖴 IntroGettingStarted - PSoC Designer - [rtcint - Read Only]						
🗹 File Edit View Project Config Build Debug Program Tools Window Help						
) 🛲   🕹 🖬 🕼   X 🖿 🕼   🗇 🌘 🚰 🚍   🔁 🖶 🔟   🗮 🗖 🐼   🖉 🖉 🖉 👘 🚸   🌑   🗰 🖇	? <b>h</b> ?					
宇宇  三 当 / 4 / 8 / 4 / 4 / 4 / 2 / 2 / 2 / 日 雪 雪 雪 冊 ● 相 1 7 7 (2 / 0)						
mone.						
<pre>IntrodetingStar Source Files boot.am DAC8.am PSoCConfit PSoCC</pre>	Qurrent:         Image: Constraint of the second secon					
branch1:						
Files Output pop a ;restore a irom stack	RAM Bank 0 Bank 1 Flash					
★ Halted Step into Halted Step out Halted Running Halted	Value Oxfe					
III III IIII IIIII IIIIIIIIIIIIIIIIIII						
For Help, press F1 A: 01 X: 14 SP: 16 PC: 0112 F: 00 Halted Idle Connected NUM						
🏽 Start   🗹 🍪 🖏 🚫 🍇 🚺 🖿 🐁 »   🚫 R. 🗞 H. 🖻 M. 🗐 M. 🗐 L. 🖓 N. 🔚 I 🗐 C. 🎛 u. 🕱 E.	. 😡 🖷 🌔 🛒 🖳 🗑 🜠 🍑 Š 7:52 AM 🦯					
	( PERSONAL ) ACCESS ) ENTERPRISE ) METRO					



#### Viewable Items

#### Access to I/O Registers, RAM, Flash, CPU registers

• CPU Registers – PC,SP,X, A, Flags

#### Bank 0 and Bank 1 viewable and modifiable

At this point in our tutorial we can view the I/O registers. The results of the "OutputV" variable will be output onto Port 2 data line, which is located at I/O Address Register 008.

In the right frame of the Debugger subsystem, click the Flash (Bank0) tab and go to memory location 008. This is the Port 2 data line, which will be output to the Pod LED display. At this point, the LEDs should be lit representing this value.



Viewable Items

### **View RAM Registers**

 → View the OutputV variable in the RAM section of the Debugger. Click the RAM tab and scroll over to the address location 0E. The value is also shown in the Watch Variable window in the lower right

• The Ram values may be dynamically changed while in the debugger.



**System Verification:** 

### Is the system working correctly? NO!

- If we let the program run we can see that the LED's are not sequencing through the 8 bit DAC values.
- Let's use the latest PSoC feature, Dynamic Event Points to find our run-time error!



#### **Test scenario:**

Using test code from GettingStarted several test cases have been developed as examples to demonstrate the usage of Dynamic Event Points.

Test Case #1 – Demonstrates using the 8-Bit and 16-Bit test threads in combination, monitoring an address location, breaking on event and turning trace on. (MEM\_DA, MEM\_DB).

Test Case #2 – Demonstrates monitoring a specific assembly instruction using the Match count parameter.

Test Case #3 – Monitoring a particular line of code 16-Bit (PC) and looking at the Flag registers.

PERSONAL

ACCESS

ENTERPRISE

METRO

CORE



### **Dynamic Event Points**

Test Case #1: Demonstrating Mem\_DA, Mem\_DB and the Combinatorial Operator.

Test: Determine if OutputV ever reaches a mid range of 2f (hex)? If it does: Turn trace on and Break.

The address location for OutputV is 0eH. (This can be found by viewing the .mp file which can be found by selecting the View toolbar  $\rightarrow$ Output.)

**Expected test outcome:** 

A Break TWO LINES BELOW the mov A,outputV line of code, the address location 0e = 2F, and the Accumulator = 2F which would be correct for the program



Debugger Events		×
8 Bit Thread	👕 State Logic	16 Bit Thread
0e <=MEM_DA<= 0e Inverted = FALSE	)— AND Match Count 1 Next State 0 Break On Trace On	2f <=MEM_DB<= 2f
Disabled	Disabled	Disabled
Disabled	Disabled	Disabled
3	Disabled	Disabled
✓ Enable 8 bit Thread       Inverted         Low compare       Input Select:       High compare         0e       <=       MEM_DA       ✓         Input Mask:       iff       Input Mask:       Iff         uP RAM Data Address (DA)       ✓       ✓       ✓	State Logic Next state:      Break     Trace On     Match count:      Trace Off     External trigger     Combinatorial Operator:     NAND     OR     OR     OR	Enable 16 bit Thread Inverted Low compare Input select High compare 002f <= MEM_DB <= 002f UP RAM Data Bus (DB)
Help	OK Cancel	



### Dynamic Event Points: Test Case #1-results

#### Test Case #1: We are looking to see if we hit a midrange of the DAC.

**Results:** If we were to hit OutputV = 2f the program would break.

It doesn't! Let's try something else.

Is the Decrement instruction working?

Debu	igger Events		×
	8 Bit Thread	👕 State Logic	16 Bit Thread
٥	0e <=MEM_DA<= 0e Inverted = FALSE Mask: ff	-D-AND Match Count 1 Next Sta Break On Trace On	2f <=MEM_DB<= 2f
1	Disabled	Disabled	Disabled
2	Disabled	Disabled	Disabled
з	Disabled	Disabled	Disabled
	Enable 8 bit Thread	State Logic     Next state:     Break	Enable 16 bit Thread
þ	linput Select:	High compare 0e Match count: Trace 1 Extension	On         Low compare         Input select         High compare           Off         002f         <=         MEM_DB         <=         002f
	If FIAM Data Address (DA)	Combinatorial Operat D− <sup>AND</sup> D− <sup>NAND</sup> D− <sup>NAND</sup> S→ <sup>OR</sup>	or UP RAM Data Bus (DB)
	Help	OK C	Apply Clear All
		PERSONAL ACCESS	CORE CORE



### **Dynamic Event Points**

Test Case #2: Demonstrate (IR) Instruction Register, and the Match Count

Test: Let's see if the DEC instruction is working correctly? When the decrement instruction (DEC) is executed 3 times Break Turn trace OFF.

**Expected test outcome:** 

A Break TWO LINES BELOW the DEC command.



44

#### Dynamic Event Points: Test Case #2

Deb	ugger Events			×
	8 Bit Thread	👚 State Logic		16 Bit Thread
٥	7a <=lR<= 7a Inverted = FALSE Mask: ff	Match Count 3 Break On Trace Off	Next State 0	Disabled
1	Disabled	Disabled		Disabled
2	Disabled	Disabled		Disabled
3	Disabled	Disabled		Disabled
	C Enable 8 bit Thread  Inverted  ow compare Input Select: High compare 7a C IP C I I	State Logic Next state: 0 Match count:	Break Trace On Trace Off	Enable 16 bit Thread
	uP instruction register (IR)	3	External trigger	uP Instruction Register (IR)
	Help	ОК	Cancel	Apply Clear All
				( PERSONAL X ACCESS X ENTERPRISE X METRO

CORE



### Dynamic Event Points: Test Case #2-results

# Test Case #2: We are looking to see if the decrement command is cycling

#### **Results: Do we hit a breakpoint relatively quickly?**

Yes! So we know that our cycle should be working

Let's try something else.

Debugger Events		<u>×</u>
8 Bit Thread	😷 State Logic	16 Bit Thread
7a ≪≡R≺≃ 7a Inverted = FALSE Mask: ff	Match Count 3 Next State 0 Break On Trace Off	Disabled
Disabled	Disabled	Disabled
Disabled 2	Disabled	Disabled
Disabled 3	Disabled	Disabled
Enable 8 bit Thread      Enable 8 bit Thread      Low compare     Input Select:     High compare     Input Mask:     If      uP instruction register (IR)	State Logic Next state:	Enable 16 bit Thread      Invetted      Low compare Input select High compare      007a c= IR v c= 007a      uP Instruction Register (IR)
Help	OK Cancel	Apply Clear All



### **Dynamic Event Points**

#### Test Case #3: Let's revisit what this program should do!

#### We know from the previous tests the following: The mid range never is hit(2F), But - the DEC command seems to be working.

## Dynamic Event Points will enable a more complex testing to find our problems: Hint there are two!





**Dynamic Event Points** 

**Test Case #3: Let's start with a simple code inspection first.** 

Why isn't OutputV ever hitting the high end?

Answer: A code review of main.asm and RCTINT.asm shows that we start with 255 and then we DEC first. So our effective range is only 254.

OK that's one bug, Now why isn't it cycling?



Let's perform a conditional branch test. (Many regulated environments require this type of testing)

Let's look and see if Branch 1 is ever executed?

Also let's look to see if how often the resetting of OutputV occurs. It should only occur after the completion of the loop 255-0.

Look at the .lst file (in the output section) for the PC values. Let's investigate the conditions of the jz command as well.



Example List file (Your's may be slightly different- do a Control Find for RTCINT)

0054) mov A,[outputV] ;if voltage variable reaches 0 reset to maximum value

011A: 51 0E MOV A,[outputV]

(0055) jz branch1

011C: A0 04 JZ 0x0121

(0056) mov [outputV],0

011E: 55 0E 00 MOV [outputV],0

(0059) branch1:

(0060) pop A ;restore A from stack

0121: 18 POP A



ebug	gger Events 8 Bit Thread	T State Logic	16 Bit Thread
	00 <=BITFIELD<= 00 Inverted = FALSE Mask: 20	D-AND Match Count 1 Next State 1 Break On Trace On	11c <=PC16<= 11c ▲ Inverted = FALSE
1	00 ≺=BITFIELD≺= 00 Inverted = TRUE Mask: 20	AND Match Count 1 Next State 2 Break On Trace Off	11c ≺=PC16<= 11c Inverted = FALSE
2	Disabled	Match Count 1 Next State 1 Break On Trace Off	11e ≺=PC16<= 11e Inverted = TRUE
3	Disabled	Disabled	Disabled
Low Do	Enable 8 bit Thread Compare Input Select: High com <= BITFIELD <= 00 Input Mask: 20 FIELD Help: Bits 0-7 0 (0x01) RAM read flag (active low) cmp Hi= cmp Lo= 00 1 (0x02) RAM write flag (active low) cmp Hi= cmp Lo= 00	State Logic ed pare Match count: Trace On Match count: Trace Off Combinatorial Operator: NAND NAND OR OR	Enable 16 bit Thread Low compare Input select High compare 011c <= PC16 <= 011c UP Program Counter (PC16)
	Help	OK Cancel	Apply Clear All



#### **Expected Results for our tests:**

We are looking in the first sequence for a false on the jz command. If it is false then the code will drop to the next command and reset the OutputV value.

Our next event is looking for the inversion of the above. Does the jz ever hit true?





### **Expected Results for our tests:**

- Let's let it run and see what happens.
- We should get lot of breaks if the code is cycling correctly.....
- Answers will be given in the class..... Let's Hit RUN!





CORE