Electronic Design

"Why Didn't I Think of That?"—Seeing Inside Embedded Systems

Micrium founder Jean Labrosse explains how developers can use debugging hardware to visualize the state of embedded systems with little or no CPU intervention.

mbedded software developers are quite familiar with using a code editor, a compiler, linker, debugger, and, of course, an evaluation board. Most of the time, these tools are all you need to develop and debug an embedded system. But what do you do when you want to verify the operation of dynamic systems like motor control, process control, chemical processes, flight systems, and more?

Modern processors have specialized debugging hardware that allows tools to display or change memory locations while the target is running. Let's explore how such debugging



1. How do you see inside a "black box" device like a microcontroller?

hardware can be used to help you visualize the state of your embedded system with little or no CPU intervention, and while the target is running.

If you've been designing embedded systems for a while, you know how complex devices have become and how hard they are to debug. Microcontroller units (MCUs), self-contained devices (black boxes) with on-chip memory, are packed with literally hundreds or even thousands of registers that are used to control the operation of various peripheral devices (*Fig. 1*).

Every toolchain comes with a debugger, which, at a minimum, allows you to stop the target and examine variables and I/O registers (in the watch window) (*Fig. 2*). Although quite useful when debugging algorithms have no real-time

additional operations, you'll need either more LEDs, or you'll have to be creative with the ones you have, e.g., blip patterns, blink rates, etc.

no-go status. However, if you want to verify the status of

7-SEGMENT DISPLAYS

Low-cost embedded systems might be equipped with either LED or LCD 7-segment displays for use by the end user (*Fig. 3*). The embedded developer can borrow the display during development to provide an indication of what's happening in the embedded system.

A 7-segment display can display numeric values in binary, decimal, hexadecimal, or even limited alphanumeric values.

component, this capability is somewhat useless when you can't afford to stop the target, e.g., motor control, process control, etc.

To monitor the proper operation of a running embedded system, developers typically revert to a number of techniques that require code to be added to the system being monitored:

LEDS

Developers of embedded systems typically have access to at least one LED they can use to indicate that something is working; when the light turns green, the CPU made it to main() or some other place of interest. LEDs are great at indicating go/ You're typically limited to the range of values you can display based on the number of digits available. Also, if you want to display different values, then you'll need a way to cycle through them. If your embedded design doesn't require a display, then you might add a display just for testing purposes. Of course, for this to work, you'll need to write code for this purpose.

CHARACTER MODULES

Character modules (LEDs or LCDs) are fairly low-cost devices that you can use as a debugging tool (*Fig. 4*). Modules are available that interface through either a parallel port (requires 6 to 10 output lines) or through a serial interface (typically UART). Character modules are available in a 1 \times 8 (1 line by 8 columns) configuration all the way to 4 \times 40. These displays are easy to use and allow you to display alphanumerical characters.

As with the 7-segment display technique, you'll have to write code to format and position the variables of interest, and program a way to select different values if the selected display doesn't have enough characters for your needs. Character modules have the added benefit of being able to display bar graphs. A chapter in Embedded Systems Building Blocks (see Bibliography) provides a longer explanation of character modules.

PRINTF()

The printf() function is, in my opinion, one of the most overused and problematic tools you can turn to. Whenever you want to display the occurrence of an event or display the value of variables, you have to format a string, rebuild your code, download it, and restart your application. The printf() outputs are generally sent to a debugger text console, an RS-232C port, or USB. Values "fly off" the screen once you reach the number of rows you can display, which oftentimes is more annoying than useful. Not only does printf() require a fair amount of code, it negatively affects the timing of your system.

GRAPHICAL DISPLAY

If your end product contains a graphical display, then you



3. Low-cost embedded systems are often equipped with LED or LCD 7-segment displays for development.

pression	Type	value	D=f=-1++92	
appTaskDisplayTCB	OS_TCB	0x2000b7dc	Decimal:83	
StkPtr	CPU_STK *	0x2000abfc	Hex:0x53	
ExtPtr	void *	0x0	Octal:0123	
NamePtr	CPU_CHAR *	0xb944	Binary:0b101001	1
StkLimitPtr	CPU_STK *	0x2000aa98		
NextPtr	OS_TCB *	0		
PrevPtr	OS_TCB *	0		
TickNextPtr	OS_TCB *	0x2000b9ec		
TickPrevPtr	OS_TCB *	0x2000b88c		
TickListPtr	OS_TICK_LIST *	536919560		
StkBasePtr	CPU_STK *	0x2000aa98		
TaskEntryAddr	OS_TASK_PTR	0x87dd		
TaskEntryArg	void *	0x0		
PendNextPtr	OS_TCB *	0		
PendPrevPtr	OS_TCB *	0		
PendObjPtr	OS_PEND_OBJ *	0		
(x)= PendOn	OS_STATE	0 ('\0')		
(x)= PendStatus	OS_STATUS	0 ('\0')		
(x)= TaskState	OS_STATE	1 ('\001')		
(x)= Prio	OS_PRIO	5 ('\005')		
(x)= BasePrio	OS_PRIO	5 ('\005')		
MutexGrpHeadPtr	OS_MUTEX *	0		
(x)= StkSize	CPU_STK_SIZE	128		
(x)= Opt	OS_OPT	3		
(x)= TS	CPU_TS	0		
(x)= SemID	CPU_INT16U	0	-	
(x)= SemCtr	OS_SEM_CTR	0		
(x)= TickRemain	OS_TICK	1		
(x)= TickCtrPrev	OS_TICK	0		
🖻 🕞 RegTbl	OS_REG[1]	[0]		
(x)= FlagsPend	OS_FLAGS	0		
(x)= FlagsRdy	OS_FLAGS	0		
(x)= FlagsOpt	OS_OPT	0		
(x)= SuspendCtr	OS_NESTING_CTR	0 ('\0')		
(x)= CPUUsage	OS_CPU_USAGE	0		
(x)= CPUUsageMax	OS_CPU_USAGE	35		
(x)= CtxSwCtr	OS_CTX_SW_CTR	3442		
(x)= CyclesDelta	CPU_TS	2396		
(x)= CyclesStart	CPU_TS	3328033009		
(x)= CyclesTotal	OS_CYCLES	0		
(x)= CyclesTotalPrev	OS_CYCLES	0		
(x)= SemPendTime	CPU_TS	0		
(x)= SemPendTimeMax	CPU_TS	0		
(x)= StkUsed	CPU_STK_SIZE	45		
(×)= StkFree	CPU_STK_SIZE	83		
(x)= IntDisTimeMax	CPU_TS	455		
DbgPrevPtr	OS_TCB *	0x2000b72c		
DbgNextPtr	OS_TCB *	0x2000b67c		
DbgNamePtr	CPU_CHAR *	0x6730		
(x)= TaskID	CPU_INT16U	0		

2. A debugger enables the developer to stop the target and examine variables and registers in a watch window.

can use it during debug to display large amounts of data and even graphs. However, the debug code would eventually need to be thrown away or hidden in the released version of your code. A graphics library requires tens to hundreds of kilobytes of code space and a lot of RAM (depends on the display resolution), consumes CPU cycles, and adds complexity to your application. There are better choices.

The above options are inadequate if you're trying to display a large amount of data or, worse yet, you forgot to include some critical value that needs to be displayed for a process-control application. You then have to edit/compile/download and run a new build, and bring your application in the "state" you are trying to observe. Also, displaying data is fine, but what do you do if you also need to change the value of variables such as setpoints, limits, gains, offsets, etc.?

GRAPHICAL LIVE WATCH

Advanced processor cores like the ARM Cortex-M or Renesas RX are equipped with a hardware debugger port that provides direct access to memory and peripherals without requiring any CPU intervention. In other words, memory and I/O locations can be displayed, or changed, at run-time without having to write a single line of target code.

The tool, called μ C/Probe, uses the debugger port found on Cortex-M or RX MCUs. It allows you to display or change the value of variables or I/O port registers while the target is running. You're able to display values by assigning them to graphical objects such as gauges, numeric indicators, LEDs, thermometers, etc. You also can change the value of variables by assigning those variables to sliders, switches, numeric inputs, and more. In addition, μ C/Probe can interface to the target via RS-232C, TCP/IP, or USB, but this requires a small target resident monitor. A <u>Segger</u> J-Link is by far the most convenient and least intrusive option.

Figure 5 shows how μ C/Probe works:

1. Write code using any editor, compile it, and link it.

2. Connect the debugger to the target debug port through, in this case, a Segger J-Link.

3. Download code to the target MCU either into flash or RAM. Then, instruct the debugger to run the code to start testing.

4. μ C/Probe reads the executable and linkable format file generated by the compiler and extracts the name of the variables, their data types, and physical memory locations (i.e., their address). Then it creates a symbol table that's used to assign variables to a graphical objects library built into μ C/Probe.

5. Drag and drop graphical objects (gauges, LEDs, sliders, etc.) and assign them to variables from the symbol table. μ C/ Probe also knows about the names and addresses of I/O ports through chip definition files (CDFs) that are built into it. This allows the user to look at raw analog-to-digital converter (ADC) counts, update digital-to-analog converters (DACs), look up or change the value of GPIO ports, and so on.

6. Once variables or I/O ports are assigned to graphical objects, press the μ C/Probe "RUN" button and the tool will start requesting (as fast as the interface allows it) the current value of those variables and I/O ports through the J-Link



4. Developers can use low-cost LED- or LCD-based character modules as a debugging tool.

stop the target, nor edit application code, compile, download, and restart.

THE μ C/PROBE IN ACTION

Let's explore an example use of μ C/Probe. How can one observe the intermediate values of a proportional-integralderivative (PID) loop where the update rate of the loop occurs at 1 kHz? As shown in *Figure 6*, μ C/Probe has a built-in 8-channel digital storage oscilloscope.

Once more, there's no need to stop the target. If the variable is available in the symbol browser, it can easily be assigned to one of the channels. It's able to trigger on the positive or negative slope of any channel, delay trigger, do pre- or post-triggering, zoom in and out, and more. Without μ C/Probe, a developer would have to scale and output the variables to available DAC ports (assuming there are some) to observe those signals. This would be highly intrusive, and you might have to rebuild your application each time you want to look at different traces.

The embedded target can run bare-metal code or work in conjunction with a real-time operating system (RTOS) kernel. μ C/Probe has built-in kernel awareness for popular RTOSs, and

interface. J-Link converts those requests into either the memory reads or writes that occur concurrently while the CPU is executing the target application.

To monitor the value of additional variables, simply stop μ C/Probe, add the graphical objects, assign them to the desired variables, press RUN, and the tool displays—or allows you to change—those variables. There's no need to



5. Using the µC/Probe tool, developers can debug embedded designs through a Graphical Live Watch window.

	- C.A.	interest in the																			
	Are loop	Acres 1	Depised 1	www	~~~~	www	XXXXX	~~~	~~~	<u> </u>	www	~~~~~	~~~~~	~~~~~	~~~~	~~~~	VYXXXX	ww	tel herb	a hain	1.00
-	FLZ.MG	11.549	14.165																1203 (4.5	10 MH	
				2.004		_										Ant Squar South	Sanghi				
																And in case					
			40			AAAAA				AAAAA		AAAAAAAA				Income	in a A A A A				
					HAAAA		AAAAA					INA BABARA			RAAAA	*****	AAAAAA	AAAA			
			.117.64													and and in the local division of the local d		12.	66 55.0	6 172	P
			۰ I		ll Ma		11111		.111	0034		MA I		±030088	//			M			
				ARRA.	hhin.	MANI		Mm			.A.U.U.	uuuumu	naaaaa		amAill	unnnu v		NAA			
				¥900	AN UV	- 11	HH	1.		10001	- A I II II		FURIER'S -	*****		HI HIYY	****	HII			
	-		{}				1111		-4			11.	11111			WH.	- 1111	1 · · · ·			
			۶ I																		
			048																		
																OTINACE 12					
			-16.78													Officers)					
			-56.7M													effision 2		<u>, </u>			
			-56578 -566367													Lucia Lucia Lucia					
			-38.78													na.a.a.r waaaan					
	01.50 01.20 01.20		-33379 -34337 -43394 -43454			 										r.a.c.a.r 19444aa			5.555 0.5	1 1 1	
	01.00 04.00 04.00	4019. 	-3679 -3620 -4239 -4239			 									 www	1.849.941 1.849.941 1.949.949 1.99			2.005	00 100	
	01.00 04.00 04.00		-36279 -36237 -42294 -42294			 www.				 www					 	алиан 12 Занаста Г. П. Д. П. Д. П. Г. Даниала (1473) 15					8
	01.00 04.00 04.00 04.00		-3639 -3630 -4239 -4242												 www	90140213 100460000000000			2.002 6.0	- 1	8
	01.00 04.00 04.00 07.00 00000000	40.0%	-90.00 -0.000 -0.00 -0.00 -0.00 -0.000 -0.00 -0.00 -0.00 -0.00 -0.00 -0.00 -0.00 -0.00 -0.00 -0.00 -0			Max / Mat. 1100					·····				 www	90140213 20140213 20140218 100402080 100402080 100402080 100402080 100402080 100402080 100402080 100402080 100402080 100402080 100402080 100402080 100402080 100402080 1004000 1004000 1004000 1004000 1004000 1004000 1004000 100400000000			0.0005 6.0	60 800	6
		4000 (340) (-60.09 -00.00 -40.00 -00.00 -00.00		2000 2000 2000 2000 2000 2000	Max / Mai. 1100 1200 1200						Descel of the second se	Landar Landar Sanglag Cack			Entraça 13 Entração 13 Entração 18 Antecesto				02 200	3
		June June June Jynes yngerthest fa			194 192	Max / Mdl.	111 Level 0.0000 p					According to the second	LUCEUR P			Entraça 2 Seria Grad C. Gradia Grad Gasagan Bri Mana Jan			2007 2.4	- 1	6
		Jane			2000 2000 2000 2000 2000 2000 2000 200	Max / Mat 11.00 12.00	11g Level 0.0000 a 0.0000 0.0000			Carrow 1 0000 1 0000	CTAR CTAR	Jacobia Contraction Jacobia C	LINNER LINNER Semple Call Deser 19 (2) Jacobier 19 (2) Jacobier 19 (2)	Contraction Contr		олиса 13 Заниса 13 Галисана, Г Данашана, Г 147.33 на			2000	- 1	3
	0100 0100 0100 0100 0100 0100 0100 010	Jane	-0.000 70.000 900.00 100.00 100.00 100 100 100 100 100		1992 1992 1992 1992 1992 1992 1992 1992	Max / Mol. 	119 CANE			Carr 1.000 1.000 1.000		Annual and a second sec	LISSEE #	Contraction of the second seco		Harrington (2) Statework (2) In Baladaria (In Bala			53005 60	- 1	3
		Arrow James Adder Adder Applique Checky Checky	 NURC <li< td=""><td></td><td>7994 1992 1992 1992 1992 1992 1992 1992 1</td><td>Max / Mail </td><td></td><td></td><td></td><td>Carr 1.000 1.000 1.000 1.000</td><td>UTUR 0.0000 p 0.0000 0.0000 0.0000 0.0000 0.0000</td><td>Access of the second se</td><td>Latinity of Lating</td><td>Contraction of the second seco</td><td></td><td>Antiversity Several Condensity (Condensity (Condensity) (</td><td></td><td></td><td></td><td>- 1</td><td>3</td></li<>		7994 1992 1992 1992 1992 1992 1992 1992 1	Max / Mail 				Carr 1.000 1.000 1.000 1.000	UTUR 0.0000 p 0.0000 0.0000 0.0000 0.0000 0.0000	Access of the second se	Latinity of Lating	Contraction of the second seco		Antiversity Several Condensity (Condensity (Condensity) (- 1	3
	or set of the set of t	Jame	 NUAR <li< td=""><td></td><td>1994 1995 1992 1992 1992 1992 1992 1992 1992</td><td>Max / Md. 15:00 -15:</td><td></td><td></td><td></td><td>Len 1.000 1.000 1.000 1.000</td><td>UTUR 0.5000 p 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000</td><td>Description</td><td>Landing Chick</td><td>Contraction Contr</td><td></td><td>aniwors Tewors Charlet Caseenen HXXX In</td><td></td><td></td><td>2000</td><td>- 1 - 4 - 100</td><td></td></li<>		1994 1995 1992 1992 1992 1992 1992 1992 1992	Max / Md. 15:00 -15:				Len 1.000 1.000 1.000 1.000	UTUR 0.5000 p 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000 0.5000	Description	Landing Chick	Contraction Contr		aniwors Tewors Charlet Caseenen HXXX In			2000	- 1 - 4 - 100	

6. The μC/Probe features a built-in 8-channel digital storage oscilloscope.

of course, as is the case with other variables, that information is displayed live (*Fig. 7*). The status of each task is displayed in a row and contains its name, task priority, CPU usage, run counter, maximum interrupt disable time, maximum scheduler lock time and—by far the most valuable piece of information from this view—the stack usage for each task.

Specifically, when designing RTOS-based embedded systems, one of the most difficult aspects is establishing the stack space needed for each task (see the blog "Detecting Stack Overflows" for more detail). μ C/Probe displays the maximum stack usage for each task using a bar graph, which provides a very quick visual indication of how close or far the stack is from overflowing. The built-in kernel awareness feature of μ C/Probe also allows developers to monitor the state of other kernel objects, such as semaphores, mutexes, queues, timers, etc.

SUMMARY

Testing and debugging real-time embedded software can be challenging. In fact, it's surprising that more tools aren't available to simplify embedded product design. Any tool that offers instant visibility into the inner workings of your application is worth looking into.

The forethought of chip designers to provide versatile debug interfaces as those found on modern processors such as the ARM Cortex-M and Renesas RX processors makes it easier for tools to peek inside running embedded systems without interfering with the CPU. Data-visualization tools like μ C/Probe let developers see inside an embedded system to effortlessly confirm the proper operation of the design, or reveal anomalies you can identify and fix, leading many to ask, "Why didn't I think of that?"

BIBLIOGRAPHY

Embedded Systems Building Blocks, Complete and Ready-to-Use Modules in C Jean J. Labrosse ISBN 0-87930-604-1 CMP Books, 2000 Detecting Stack Overflows (Part 1 of 2) Jean J. Labrosse https://www.micrium.com/detecting-stack-overflows-part-1-of-2/ March 8, 2016 Detecting Stack Overflows (Part 2 of 2) Jean J. Labrosse https://www.micrium.com/detecting-stack-overflows-part-2-of-2/ March 14, 2016

Reset Stats	A	Total CPU	100-Jsage	64	otal CPU Usage: 0	00% 100 0	% Avai Usec 70 Tota	Micrium Hi lable	AP Usage (B	o%	-	relianeo								(·µ	C/O The Real-	Time Ker
	Task(s) Performance											Task:	Stack	Task Queue					Task Semaphore				
Item	Cur Task	Name	Prio	State	Pending On Object	Pending On	Ticks Remaining	CPU Usage	Context Switch Counter	Interrupt Disable Time (Max)	Scheduler Lock Time (Max)	#Used	#Free	Size	Stack Usage	Entries	Entries (Max)	Size	Msg Sent Time	Msg Sent Time (Max)	Ctr	Signal Time	Signal Time (Max)
٥		App Task Signal Gen Ch	6	Delayed			61	0.09 %	200	32.43	0.00	43	85	128	3.59 %	0	0	0	0.00	0.00	1,323	9,511.64	16,566.7
1		App Task Buttons	4	Pending	Event Flag Group	Push Buttons Status	0	0.00 %	0	0.00	0.00	59	69	128	46 00 %	0	0	0	0.00	0.00	0	0.00	0.0
2		App Task Display	5	Delayed			0	0.13 %	100	31.36	0.00	45	83	128	5.16 %	0	0	0	0.00	0.00	0	0.00	0.0
3		App Task Start	3	Delayed			54	0.00 %	24	32.36	0.00	72	56	128	56.25 %	0	0	0	0.00	0.00	0	0.00	0.
		uC/OS-III Stat Task	6	Delayed			10	0.35 %	133	32.43	0.00	47	81	128	3 6.72 %	0	0	0	0.00	0.00	0	0.00	0.
5		uC/OS-III Tick Task	1	Pending	Task Semaphore	Task Sem	0	12.92 %	25,068	72.43	0.00	51	77	128	39.84 %	0	0	0	0.00	0.00	0	0.00	0.0
		uC/OS-III Idle Task	7	Ready			0	86.62%	24,938	27.54	0.00	19	45	64	29.69 %	0	0	0	0.00	0.00	0	0.00	0.

7. The µC/Probe has built-in awareness of popular real-time operating system kernels.