Let's Build a Robot Processing Platform

By Dr. James O. Gouge
Co-Authored by Mark E. Allred

So you want a better processor to drive your robot. There really isn't one commercially available with the processing power and memory to accomplish any high level functions.

Why not build it? The components for a powerful and flexible robot brain, or processing platform, are available, so we can build one ourselves.

What do we want our processor to do? We want to have as many independent, simultaneous processing streams or functions as we possibly can. Why? Because we have to drive our motors, our servos, our sensors, and our logic.

You can build this yourself on your kitchen table with minimal skills and tools.

We can make a robot brain that will support our servos, sensors, and memory, etc. out of a very low-cost device manufactured by Parallax called the Propeller chip.

The Propeller processor chip actually contains eight independent processors called COGs, 32K bytes of ROM, and 32K bytes of RAM as well as 32 I/O pins. This makes it a very flexible device with the best cost-performance characteristics in the industry. We can use it as our core processing unit and design and build additional logic to increase its memory to two megabytes expandable up to 16 megabytes.

Parallax manufactures a low-cost ($39.95) prototyping board which includes the Propeller chip, called the Prop Proto USB Board ("Proto board" for short) which can be easily modified to attach expansion logic and memory for our project.

This project will consist of four articles in which we will build or modify a robotic processing platform.

In this first article, we will modify a Proto board to be able to run the development debugger which we have designed and refer to as the KISS Debugger v1.0 (Keep It Simple Stupid). We will also explain the debugger.

As an intended side effect, this modified Proto board will also be prepared to attach to the memory expansion board which will be constructed and explained in the second article.

We will modify a second Proto board in the third article to support the servo driver/control software which we have designed. The software will be explained along with a general description of servo design principles.

In the fourth article, we will install a fully functional operating system software package which supports file management and general operating system commands. We will also explain how to write and install your own software commands for this new platform.

Ok. Let's begin the modification of the Proto board to support the KISS Debugger and memory expansion bus.

For this article, you will need to purchase from Parallax (www.parallax.com) the Propeller Proto Board (USB), part #32812 for $39.99 and the Propeller Proto Board Accessory Kit, part #130-32212 for $14.99. You will also

need a power supply that provides 6-9 VDC at 1.5 amps. This can be purchased from Parallax for $10.99, part #750-00009. In addition, you will need a NTSC video monitor such as a small television and a PS/2 keyboard. We are using NTSC video because it uses fewer I/O pins leaving the remaining I/O pins available for our expansion purposes.

You will need to purchase two 10-pin right angle connectors, one 2-position power block, one right angle RCA NTSC female video jack, and 12 straight wire wrap connector posts. You will need a wire wrap tool. You can buy one from Radio Shack, or it is also included when you purchase our modification kit. We also offer it standalone. Purchase three 25-foot spools of Kynar wire for the wire wrapping and a wire wrap tool.

You may wish to purchase the Cerebral Cortex Board Modification Kit from us (www.machineinteltech.com) that contains everything needed for this project including the solder, wire wrap wire, hook-up wire, a wire wrap tool, and all components necessary for this modification.

Now it's time to solder.

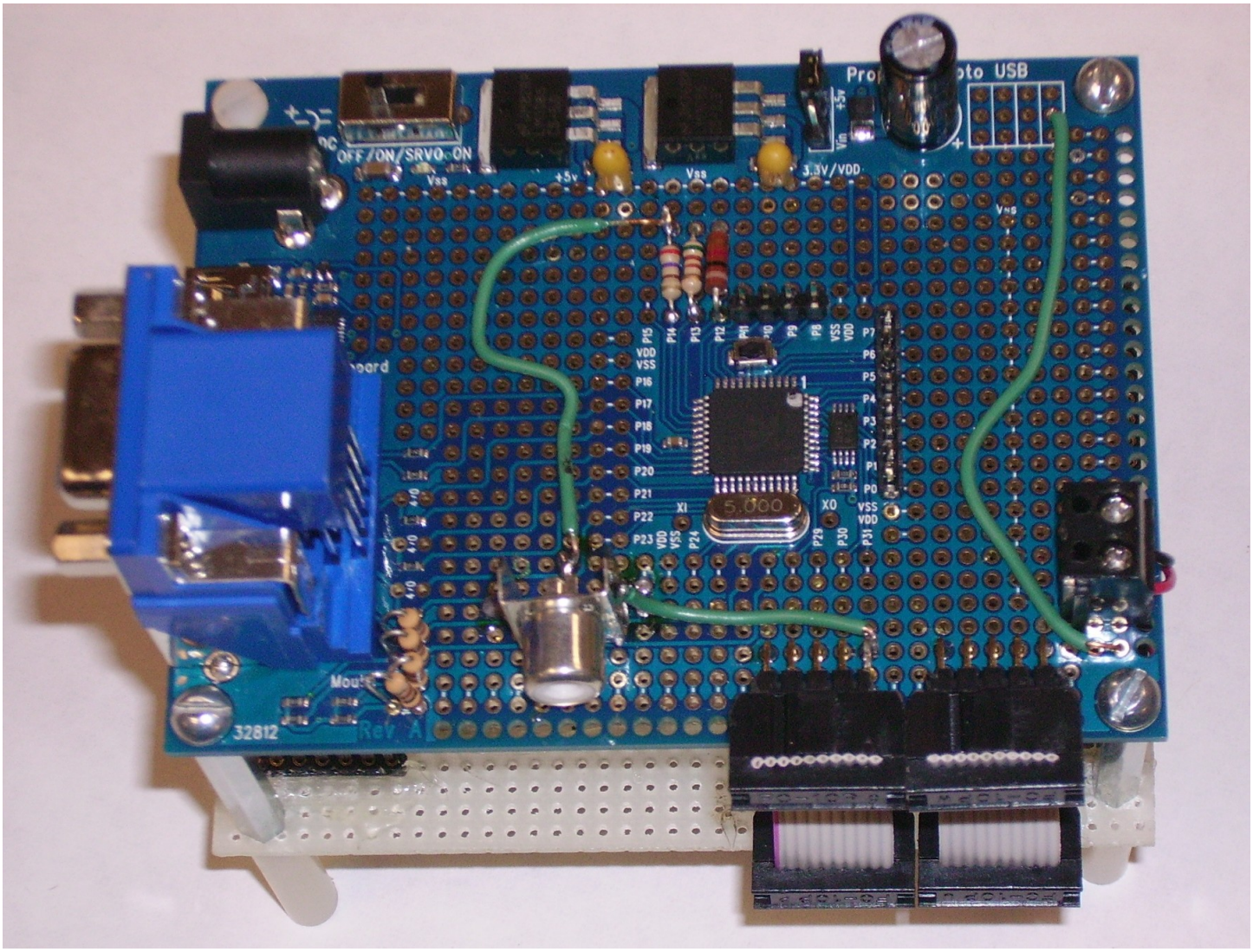
Practice soldering on some other trash board until your solder joints flow smoothly and freely. We do not want to damage the boards we are modifying. The boards are extremely sensitive. You must be careful with them. Do not subject them to harsh environments, either physically or electrically, such as excessive heat, high humidity, condensed moisture on the board, or static electricity.

As we build this platform, refer to images Proto1, Proto3, and Proto4 for the top (component) side view and image Proto2 for the bottom or wiring side view. The image Proto5 shows the modified board attached to the SRAM memory expansion board which we will build in article two.

Step 1: Plug in the soldering iron and provide a wet cloth or sponge to continually clean the soldering iron. A clean soldering iron makes for good connections. You should clean the soldering iron before soldering each component. This is good construction technique.

Step 2: Take our straight wire wrap posts and break off a group of eight, a group of four, and a group of two. The group of eight is for the eight-bit parallel bus, the group of four is for the control signals for the eight-bit parallel bus, and the group of two is for the full-duplex high-speed serial bus to interconnect Proto boards which allow multiple Proto boards (including the Propeller chips) to operate together. This is necessary to run the operating system which we will install in article four. Note that we are creating a massively parallel architecture here with multiple Propeller chips, each with eight subprocessors called COGs.

Step 3: The pins are longer on one side than the other. We will start with the eight straight pins. The eight pins go to Proto board pins labeled P0 – P7. Slide the pins through the holes provided so that the longest ends are through the bottom of the board. Ensure that the eight pins are seated during the solder procedure. Now smooth solder each pin on the back side of the board. Make sure that they flow very nicely and smooth. Neatness is important. Do NOT allow the solder to flow between pins. This can short the I/O pins and damage the Propeller chip.

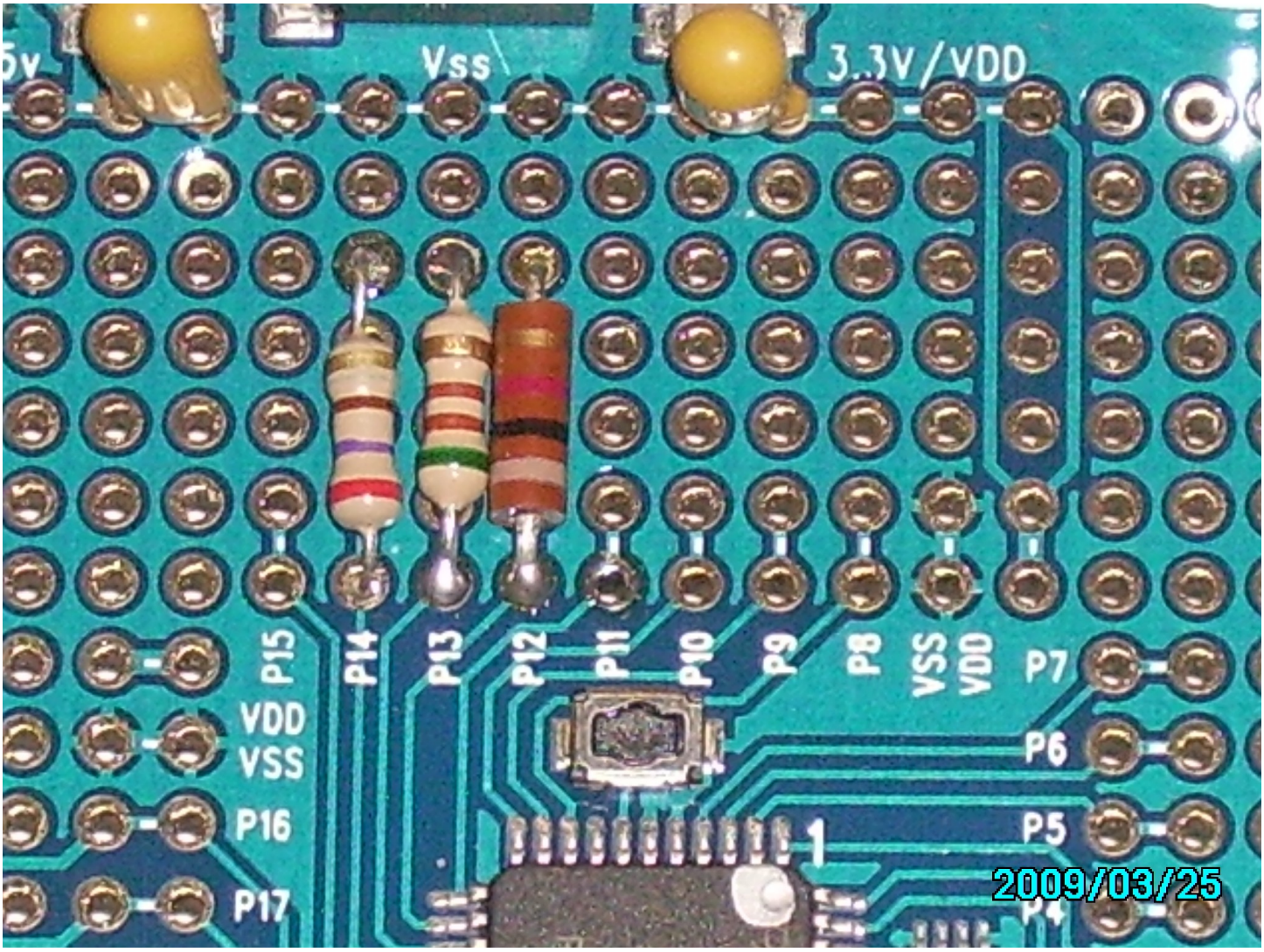


Component side of the modified Proto board.

Step 4: Insert the four straight pins into Proto board pins labeled P8 – P11. Follow the same procedures as in step three to insert, seat, and solder these pins.

Step 5: Insert the two pins into Proto board pins labeled P24 – P25. Again, follow the same procedure as above to solder. This will not be seen in the current topside photo, however P25 is next to P24, although not labeled due to the presence of the crystal.

Step 6: Locate three resistors, one 1.0k – 1.1k ohm quarter watt, one 470 – 510 ohm quarter watt, and one 220 – 270 ohm quarter watt. Prepare these three resistors by bending the leads at a 90 degree angle as close to the body of the resistor as possible. Do this for each resistor. These will be used to generate the NTSC baseband video.



Three resistors necessary for generating the NTSC baseband video.

Step 7: Insert one end of the 1.0k ohm resistor into pin P12. Solder this end. The other end goes four holes across from P12 toward the voltage regulator component marked VSS. Firmly seat the resistor and bend the lead out of the way for now. Do not solder this end yet. Cut the P12 lead of the resistor flush with the board using small diagonal wire cutters.

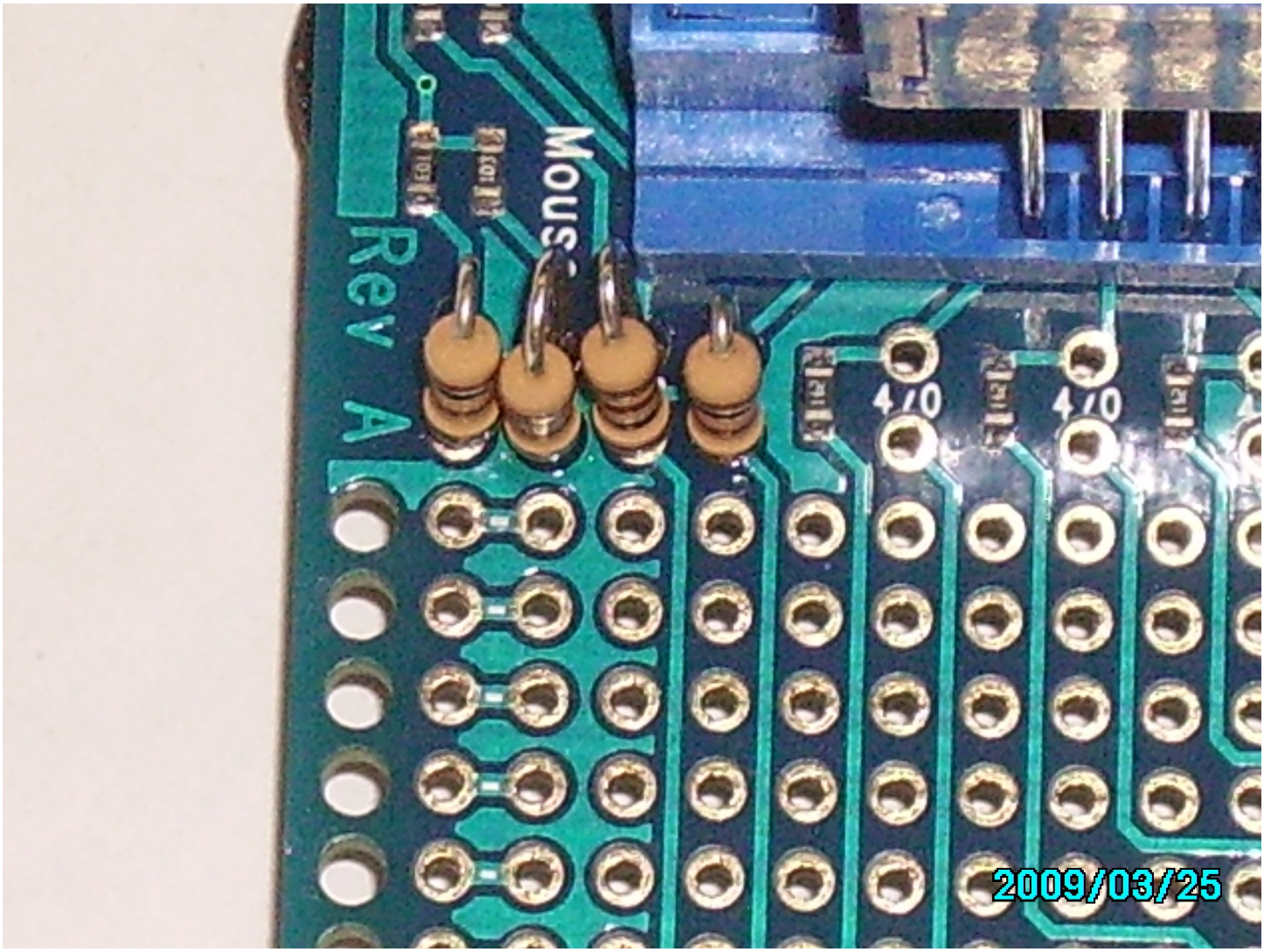
Step 8: Next insert the 510 ohm resistor into P13 and solder. Cut the excess lead flush with the board. The other end goes four holes across from P13. Bend the lead out of the way as before.

Step 9: Insert the 270 ohm resistor into P14 and solder. Cut the excess lead flush with the board. The other end goes four holes across from P14. Now turn the board over and bend the three resistor leads, the 1.0k ohm, the 510 ohm, and the 270 ohm resistor leads so that they touch each other and neatly trim off the excess. Now solder this end of all three resistors together.

Step 10: Insert the connector block from the Parallax Accessory Kit into the holes provided. This block contains VGA connector (disabled by our software), and two PS/2 ports for mouse and keyboard. The keyboard is enabled, but the mouse is disabled. Be very careful not to bend any of the leads. On the bottom side of the board, solder all of the pins going to the connector block taking care not to let the solder bleed across and short out any other pins. Should you have that problem, use the desoldering copper wire braid which can be purchased from Radio Shack, an electronics store, or from the modification kit that we sell. Place the

desoldering braid on top of the unwanted area of solder. Then place the soldering iron on top of the braid applying a small amount of pressure. Wait for the solder to melt and seep into the braid. Then remove the braid and soldering iron together. This will provide a clean connection and remove the excess solder.

Step 11: Locate four 100 ohm quarter watt resistors. Bend one lead along the body of the resistor so that it can be inserted into the board in a vertical position. See image Proto3. Insert these resistors into the holes labeled 100 in the vertical position as shown in the figure one at a time. Solder these resistors in and cut the leads off flush with the bottom of the board. Be sure to reference the image before inserting and soldering any components. These four resistors will enable the keyboard for use by the debugger and operating system.



Four 100 ohm quarter watt resistors.

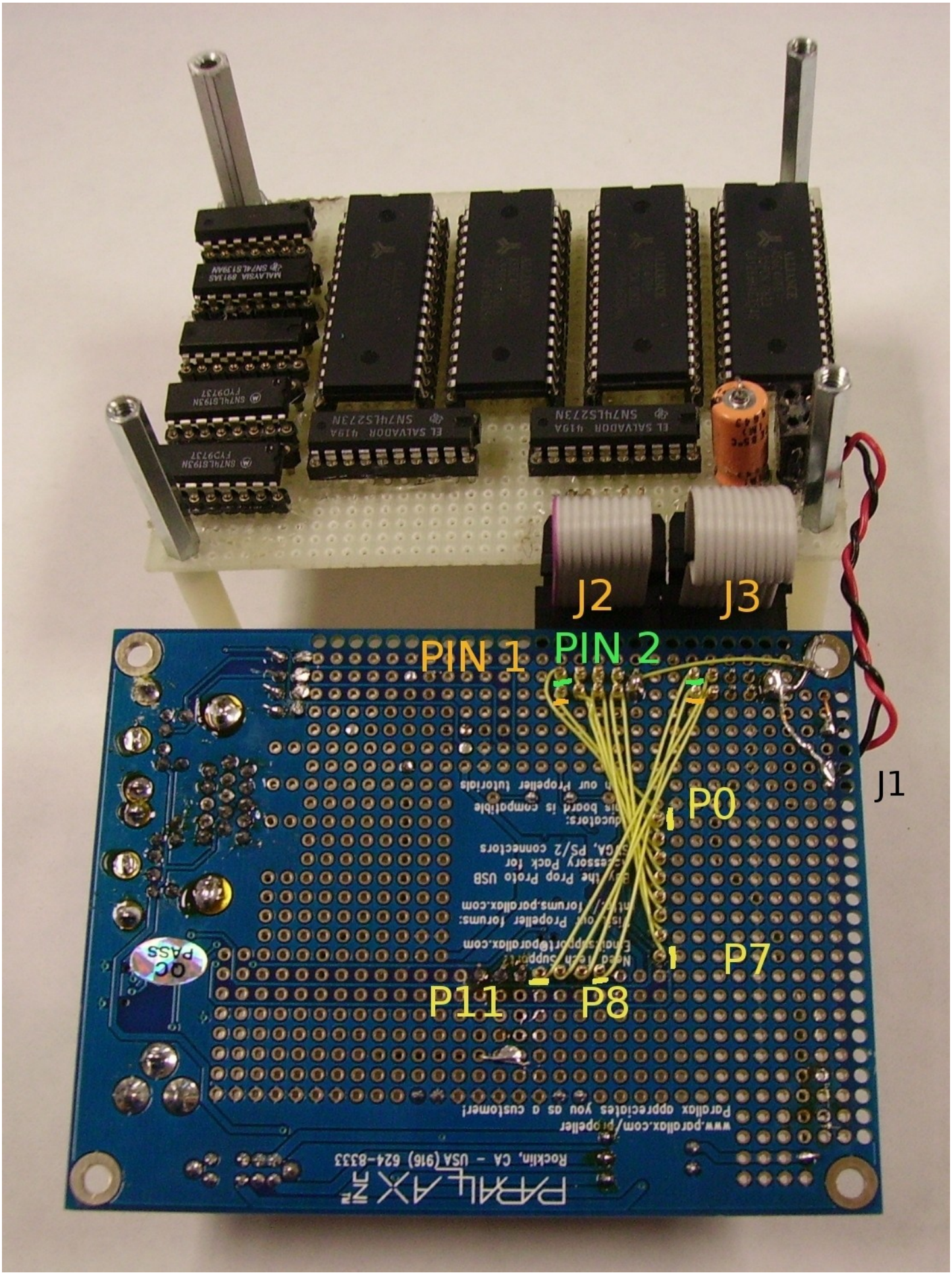
Step 12: Locate the two-position power block (J1). Insert and solder into the board exactly as shown in image Proto1. This will be used to provide power for the memory expansion board which we will build in article two. The power block should go into the third and forth holes counting from the lower right hand corner. Refer to image Proto1.

Step 13: Insert the two right-angle 10-pin connectors exactly as shown in image Proto1. Carefully reference the bottom of the board in image Proto2 for proper hole insertion. Solder the pins one, two, nine, and ten for both connectors. This will hold the connectors in place, as the four corners of each are soldered. Be careful not to use excessive solder. Be sure that your board looks identical to the images provided.

Step 14: Find, insert and tack solder the right-angle female RCA video jack on the board as shown in the images. Again, follow all precautions as previously stated. Ensure that the video connector has enough solder to be firmly attached to the board.

Step 15: Next, solder a piece of #28 hook-up wire to the junction of the three resistors as shown in the image. Solder the other end to the center conductor of the video jack which was mounted in Step 14.

Step 16: Solder a piece of #28 hook-up wire to the outer case of the RCA female video jack. Solder the other end to pin 9 of J2 as shown in the images. On the bottom side, solder a piece of #30 gauge wire to pins 9 and 10 of J2 and attach the other end to pins 9 and 10 of J3.


J2
J3
PIN 1
PIN 2
J1
P0
P7
P8
P11

This is the wire wrap side of the Proto board. Note the completed two megabyte memory expansion board next to it. The Proto board will mount on top of the memory expansion daughter board.

Step 17: Carefully solder a piece of #28 hook-up wire to the edge of the plated-through mounting hole at the corner of the Proto board where the mounting posts will be attached. This is a ground connection.

Step 18: Now solder a piece of #28 hook-up wire from pins 9 and 10 on J2 to the power block (J1) pin 1 as shown in the images.

Step 19: Solder a piece of #28 hook-up wire to power block J1 pin 2 and solder the other end to the center hole on the block of holes outlined in white on the board. This provides +5 volts to the power block J1.

Let us pause a moment for a short lesson in wire wrapping. When you use a wire wrap tool, the end result is very neat and professional. The wire wrap tool will both wrap and unwrap your wire. Also, inside the handle there is a wire stripper. It perfectly fits #30 gauge Kynar wire wrap wire. When wrapping, first use the wire stripper to strip off about a half inch of insulation.

If you look at the tip of the wire wrap tool, you should see a hole in the center. This will slip over the pin. If you look closely, there is a smaller outer hole as well that leads to a groove in the outside of the wire wrap tool. Insert the stripped end of the wire into the outer hole and thread it into the outside groove of the wire wrap tool. Now you are ready to wrap.

Take a practice pin and place the wire wrap tool over the pin at the center hole of the tool. Now place your index finger on the tip of the tool and turn the tool clockwise. Keep turning several times until the wire has been completely wrapped. Take the wire wrap tool off the pin, and you should see one-half to one wrap of insulation around the pin before the bare wire is wrapped the rest of the way around the pin. This helps avoid shorts between pins.

To unwrap, simply place the tool over the wrapped wire and pin at the center hole of the tool and turn counter clockwise with the pressure of the tip of your finger on the top of the tool. Your wire should come off very easily.

Step 20: Using the #30 gauge Kynar wire, connect a wire wrap wire from pin 1 of J2 to Propeller I/O pin P0.

Step 21: Using the #30 gauge Kynar wire, connect a wire wrap wire from pin 2 of J2 to Propeller I/O pin P1.

Step 22: Using the #30 gauge Kynar wire, connect a wire wrap wire from pin 3 of J2 to Propeller I/O pin P2.

Step 23: Using the #30 gauge Kynar wire, connect a wire wrap wire from pin 4 of J2 to Propeller I/O pin P3.

Step 24: Using the #30 gauge Kynar wire, connect a wire wrap wire from pin 5 of J2 to Propeller I/O pin P4.

Step 25: Using the #30 gauge Kynar wire, connect a wire wrap wire from pin 6 of J2 to Propeller I/O pin P5.

Step 26: Using the #30 gauge Kynar wire, connect a wire wrap wire from pin 7 of J2 to Propeller I/O pin P6.

Step 27: Using the #30 gauge Kynar wire, connect a wire wrap wire from pin 8 of J2 to Propeller I/O pin P7.

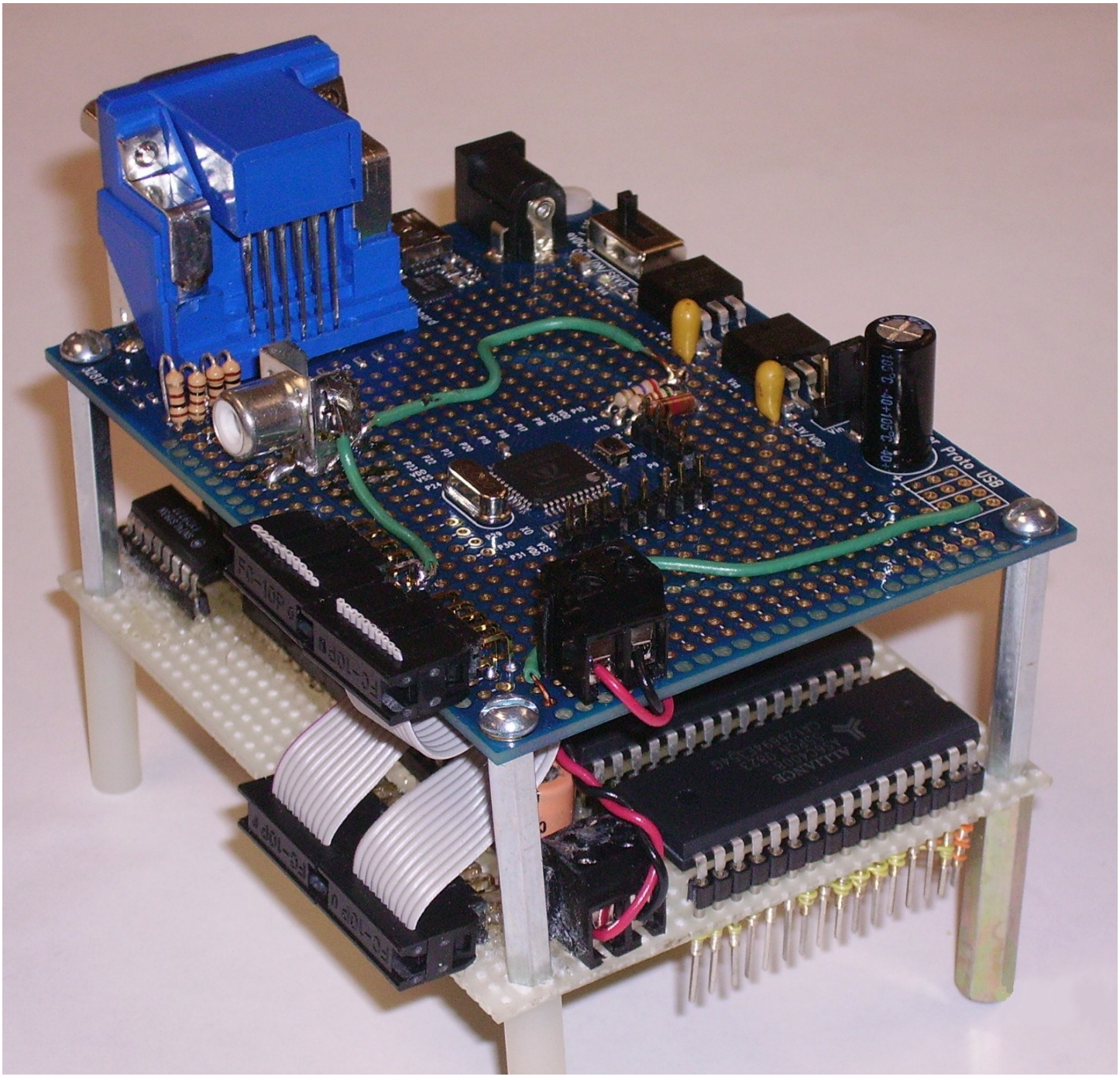
Step 28: Using the #30 gauge Kynar wire, connect a wire wrap wire from pin 1 of J3 to Propeller I/O pin P8.

Step 29: Using the #30 gauge Kynar wire, connect a wire wrap wire from pin 2 of J3 to Propeller I/O pin P9.

Step 30: Using the #30 gauge Kynar wire, connect a wire wrap wire from pin 3 of J3 to Propeller I/O pin P10.

Step 31: Using the #30 gauge Kynar wire, connect a wire wrap wire from pin 4 of J3 to Propeller I/O pin P11.

You are now ready to load and test the KISS Debugger software. You must have Parallax Propeller SPIN compiler and software library, both of which may be downloaded from Parallax (www.parallax.com). These need to be installed on your PC.



This shows a modified Proto board on top with the two megabyte expanded memory board mounted on the bottom. This now shows how J1, J2, and J3 will link to the daughter board in article two.

Secondly, you need to purchase and download the KISS Debugger from us (www.machineinteltech.com).

Now plug in the DC power block (6-9 VDC), the USB cable from your PC to the Proto board, a PS/2 keyboard to the keyboard connector, and a NTSC video display (i.e. television, etc.) to the Proto board using a standard male RCA video cable.

Turn on the power switch on the Proto board to either the ON or SRVO ON position.

Run the Parallax SPIN compiler and load the KISS Debugger (a .spin file).

Now press the F11 key to compile the debugger and download it to the Proto board EEPROM. It should then automatically start and display its opening message.

Now we will explain the operation of the KISS Debugger.

The Propeller is a system on a chip with 8 cores or processors that run in parallel. In our robot processing platform, these 8 processors or Cogs communicate with each other, with Cogs in other Propeller chips and with the outside world through a variety of I/O devices and sensors.

Sometimes you have to create your own tools in order to build something that's never been built before.

What is a debugger? And why do I need one? The **KISS Debugger** from Machine Intelligence Technologies is an indispensable tool for writing and debugging your Parallax Propeller software applications.

A debugger is a great time saver that serves as a test bed to get your project up and running quickly. This debugger is also a great resource for code to include in your own projects.

A simulator can't provide a real time development environment like the KISS Debugger. Many routines that run on a simulator won't run on the target machine. There is just no substitute for testing your code in the real time Propeller environment. This simple debugger is designed to use very little of the RAM in your development Cog while providing valuable information on what is actually happening in real time.

Most professional programmers use a debugger to help them write and debug software applications. A good debugger is a great time saver that eliminates the need to generate a user interface every time you wish to test an idea or a piece of code.

The KISS Debugger (Keep It Simple Stupid) is designed to use the smallest possible portion of Propeller resources and therefore much of the usual fluff has been omitted. Each character and each Spin/Assembly instruction requires additional memory and processor resources. In this article, snippets of the debugger code will be used as examples to explain how the debugger works and how it might be used in the development of programs for Propeller based systems.

Let's first examine the issue of a simulator versus a debugger. We'll use the following snippet of code from the KISS Debugger as an example of code that runs on a simulator but doesn't run on the target machine, i.e. the Propeller.

In the following example of self modifying code, the **WRLONG** instruction would not execute properly on the Propeller without inserting a **NOP** instruction between the **MOVD** instruction and the **WRLONG** instruction that it modifies. This **NOP** could be removed when the code is run on a simulator and the routine would still

execute properly. A simulator would not reveal this or other subtle timing issues which are the most difficult to troubleshoot.

Using a simulator in a case such as this would serve to confuse the developer (that would be you) because the code would execute just fine without the **NOP** instruction on the simulator but would not execute properly on the Propeller.

If this code, without the **NOP** instruction, were tested as a routine using the KISS Debugger, this timing problem would be readily apparent. In fact, this code snippet from the KISS Debugger executed as if the **MOVD** didn't exist until the **NOP** instruction was added, which does nothing but allow 50ns of time, 4 clock cycles, to pass to resolve the timing issue. In this case we used the debugger to debug itself.

```
************** Example Cog ASM Code Snippet ******************
               movd    :ModifyWr, CogAddrC
               nop                'this NOP resolves the instruction
                                  'pre-fetch "Read before Modify" Cog timing issue
:ModifyWr      wrlong  CogAddrV, Cog_Main_Mem_Scr_Adr    'Put Cog data from
                                                         'Cog_Addr into Cog_Scr[2]
************************************************************
```

Let's put our heads together and think this through. Since a simulator can't provide the timing environment of the target machine, it can not be used to develop timing sensitive code. When you're controlling a robot, you are using a lot of timing sensitive code!

The KISS Debugger runs on the Propeller and therefore can be used to develop both timing sensitive code and code that isn't timing sensitive. This makes it the overall best choice of tools for software development on the Propeller system. As a matter of fact, you could be far into a project, using a simulator to debug code, and then find out that the code wouldn't run on the Propeller hardware. This could force you to rethink your whole approach.

The Debugger includes the following commands:

A: Display Main RAM address of label
    See where a piece of SPIN code or data starts in Hub RAM
D: Display Main Memory Block
    Display consecutive memory locations in Hub RAM
E: Set/Clr DIRA
G: Go To Loc
H: Output this list of cmds
    Help command
L: List Cog RAM contents
   List the contents of 32-bit words in Cog RAM addresses 0 through 495 ($1EF)
M: Display/Change Main Memory Bytes
    View and optionally change the contents of memory bytes in Hub RAM
O: Set/Clr OUTA
R: Dump Regs
    List the contents of 32-bit special purpose registers in Cog RAM
    addresses 496 ($1F0) through 511 ($1FF)
S: Start Next Free Cog
T: Execute Test Code
    Insert your test code in this command. Then use this command to execute it.
X: Select & Start Cog

Select a specific Cog, download your ASM code and execute it in the Cog
Del Key: Terminate a command
Terminate a command that performs a variable number of operations
End Key: Terminate commands D, E, L, O, M
Terminate instructions that display or modify consecutive locations of memory
Esc Key: Exit Debugger

Let's take a look at the commands one by one and how they might be used.

The "A" command displays the absolute address of a routine, a variable or an array in main RAM. Using the address with the "D" command, you can display the contents of a variable, an array, a call stack, or the entry point to a routine in RAM or EEROM. The "M" command can also be used to examine or change the contents of absolute memory locations in main RAM as well, with this information. The "A" command can easily be modified to output the addresses of several items.

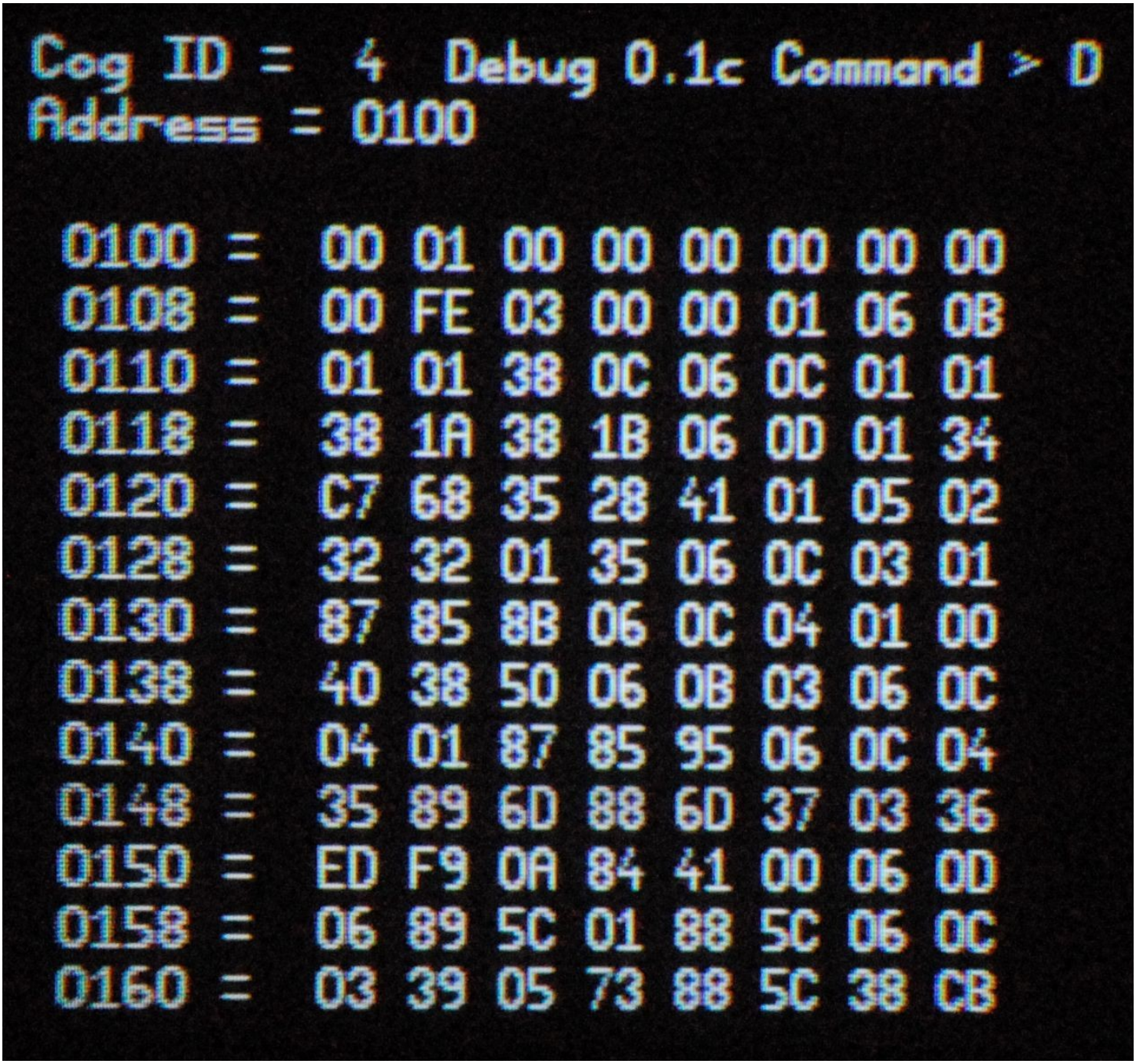


IMAGE 1

The “D” command displays a block of absolute memory locations in main RAM/ROM

This command does not change the contents of these locations.

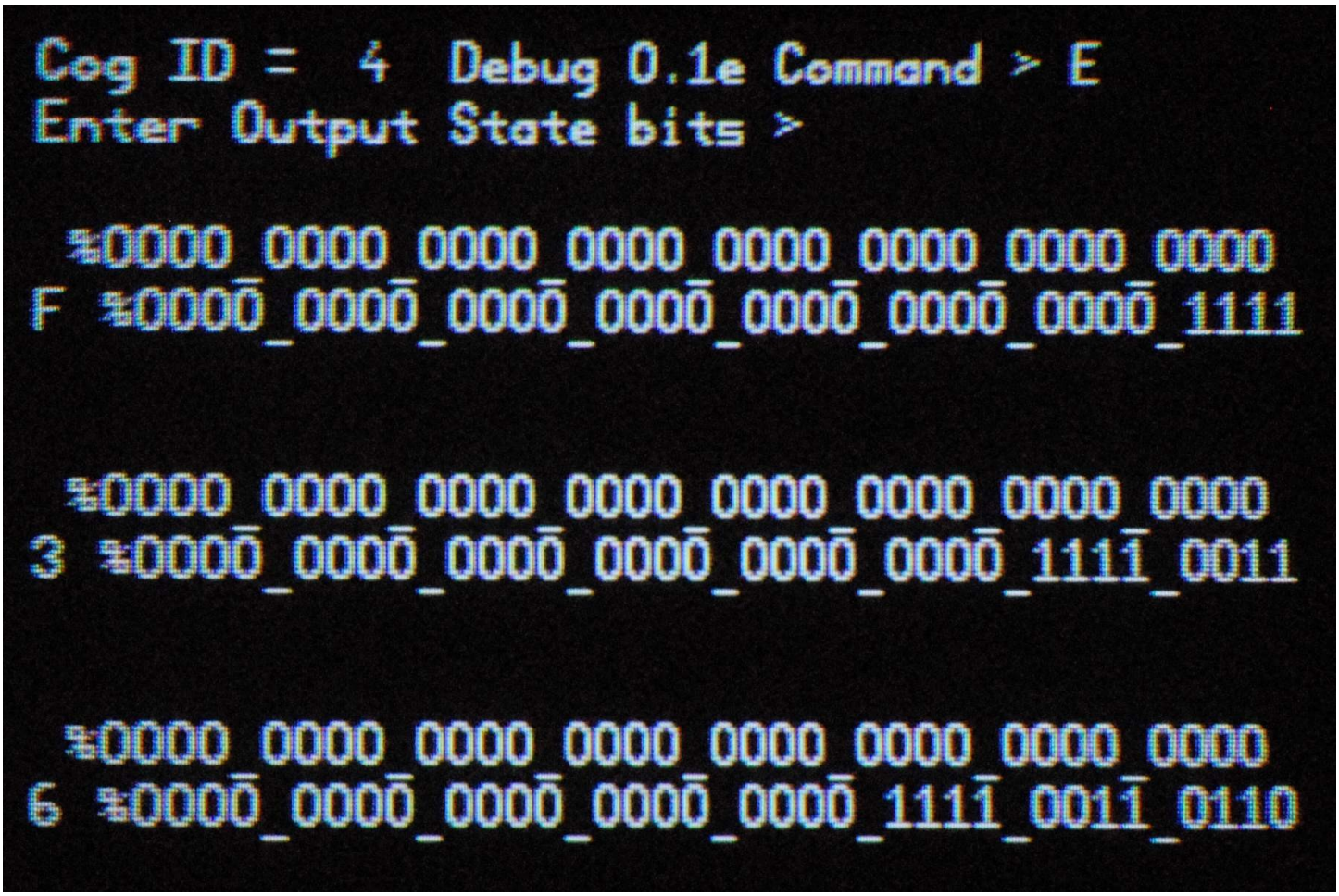


IMAGE 2
The "E" command displays or sets the "**DIRA**" register.

This command, in combination with the "O" command, allows you to test your external circuitry or connections with little more than a logic probe in most applications.

The "G" command is used to directly execute a routine in Cog RAM. It could be used to switch the Cog from performing one function to another without reloading the Cog. This should only be used by advanced programmers as it can cause the Debugger to run off into the weeds (that's a technical term) and you would have to completely reboot the Propeller. Use this command with CAUTION!

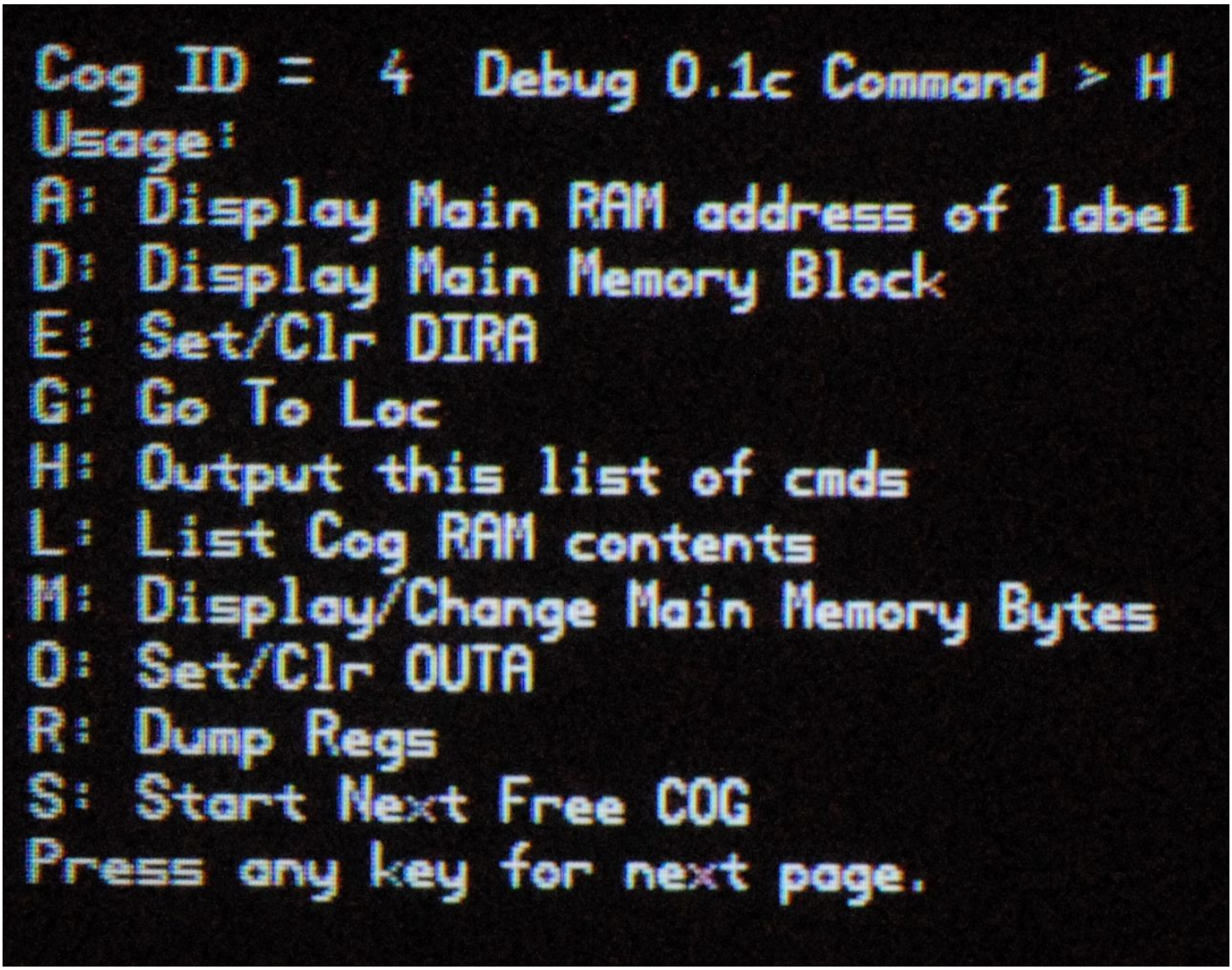


IMAGE 3
The “H” command displays the command list.

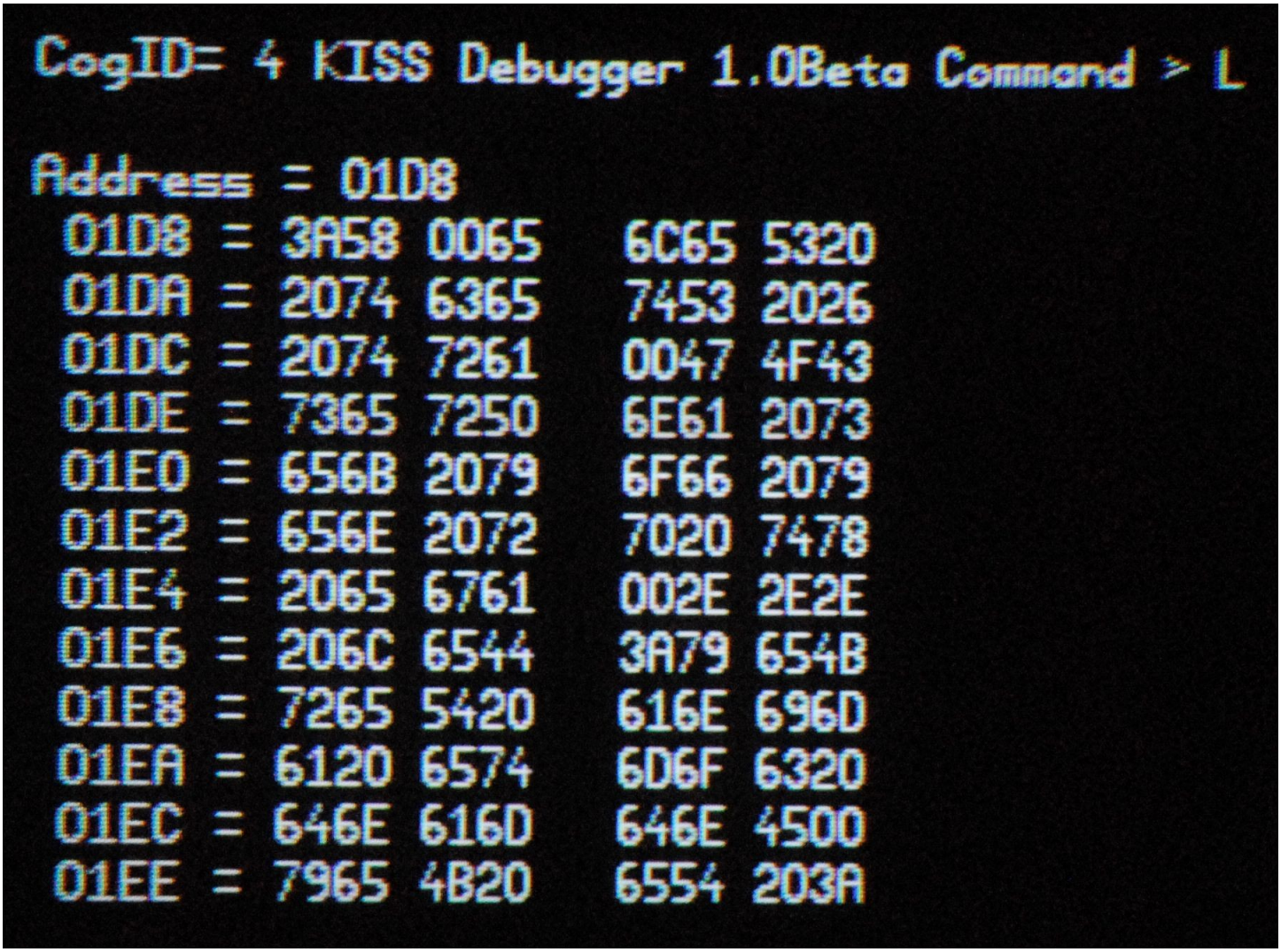


IMAGE 4

The “L” command displays Cog RAM addresses 0-495 ($0-$1EF). The current version does not allow you to change Cog RAM to keep you from shooting yourself in the foot.

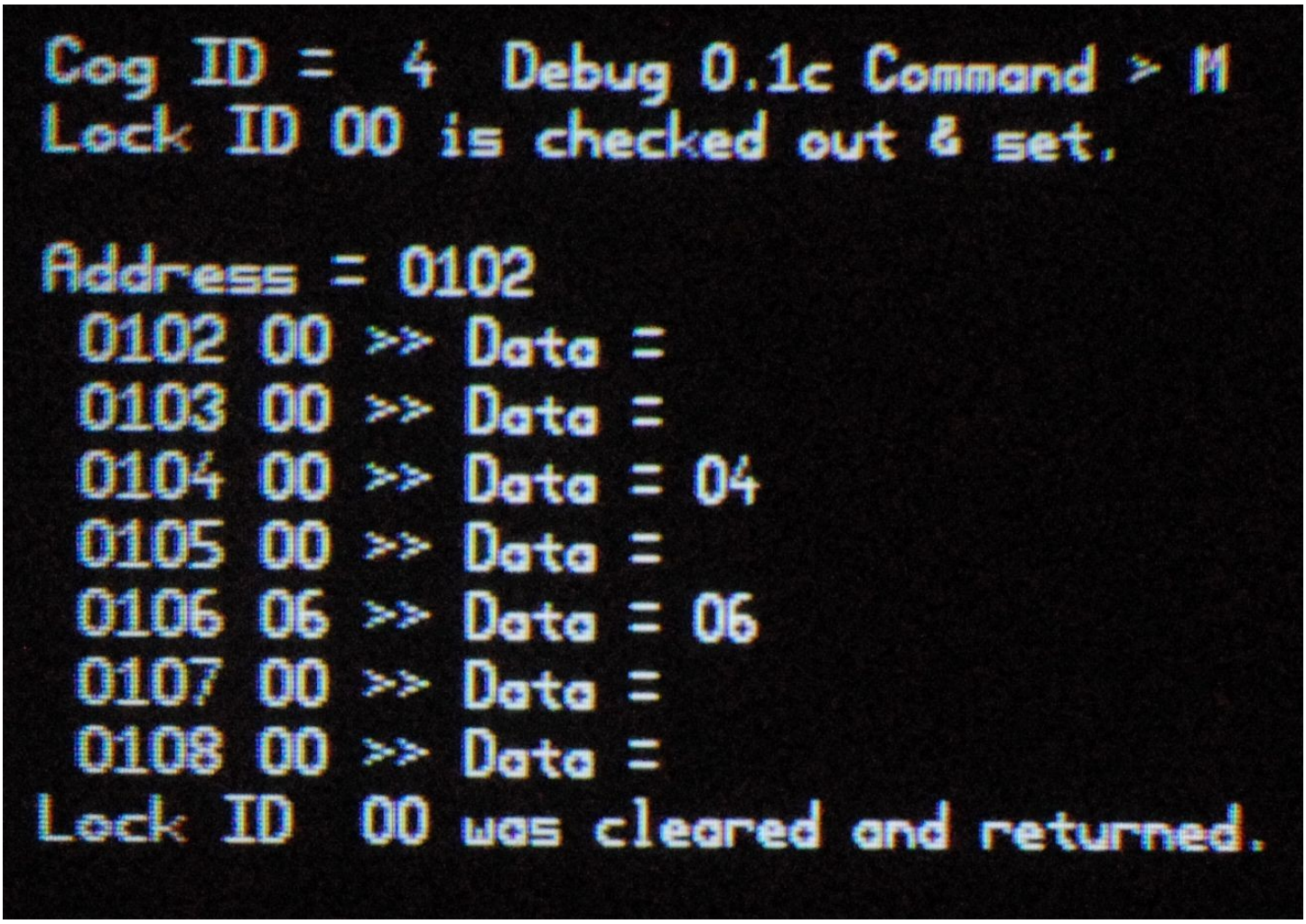


IMAGE 5
The "M" command displays and/or changes absolute memory locations.

If you don't enter a data value, the location is left unchanged.

The "O" command displays or changes the "**OUTA**" register and is normally used in conjunction with the "M" command to control an I/O pin.

The "R" command displays the 16 special registers at $1FO - $1FF. The debugger does not allow any of the registers except the "DIRA" and "OUTA" to be changed.

The "S" command starts the next free Cog and displays its number. The command returns and displays $F if no Cog is free.

The "T" command allows you to test your routines. Code segments can be inserted into the "T" command including code to output results. This is very handy for testing instructions that you may not fully understand.

The "X" command initializes and starts a specific Cog. You simply enter the Cog number. The "X" command then executes a **COGINIT** instruction to load a Cog and start it running. Make sure you select an unused Cog or

one that isn't doing something you would not want to interrupt.

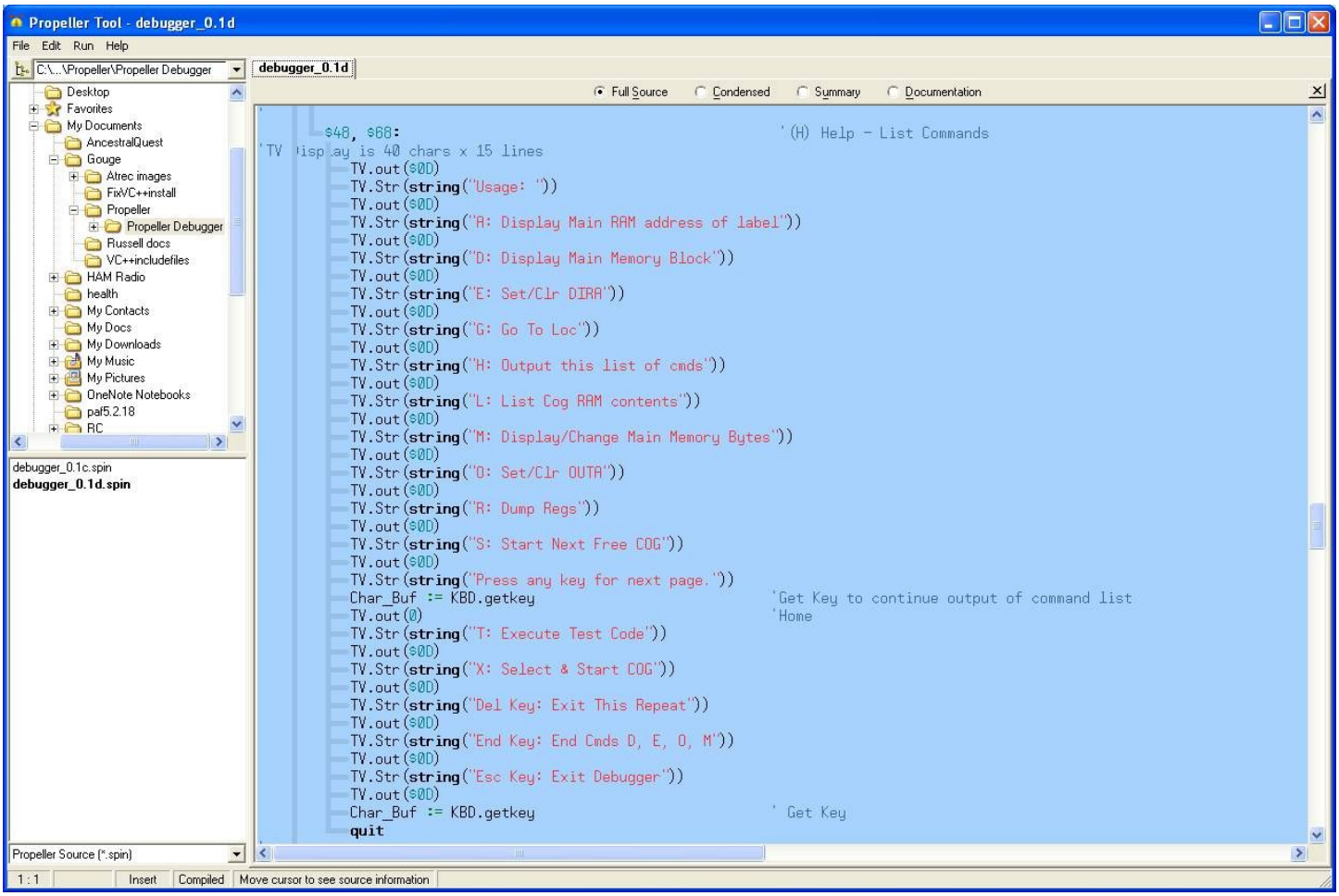


IMAGE 6

Parallax Propeller Tool displaying the KISS Debugger Help Routine.

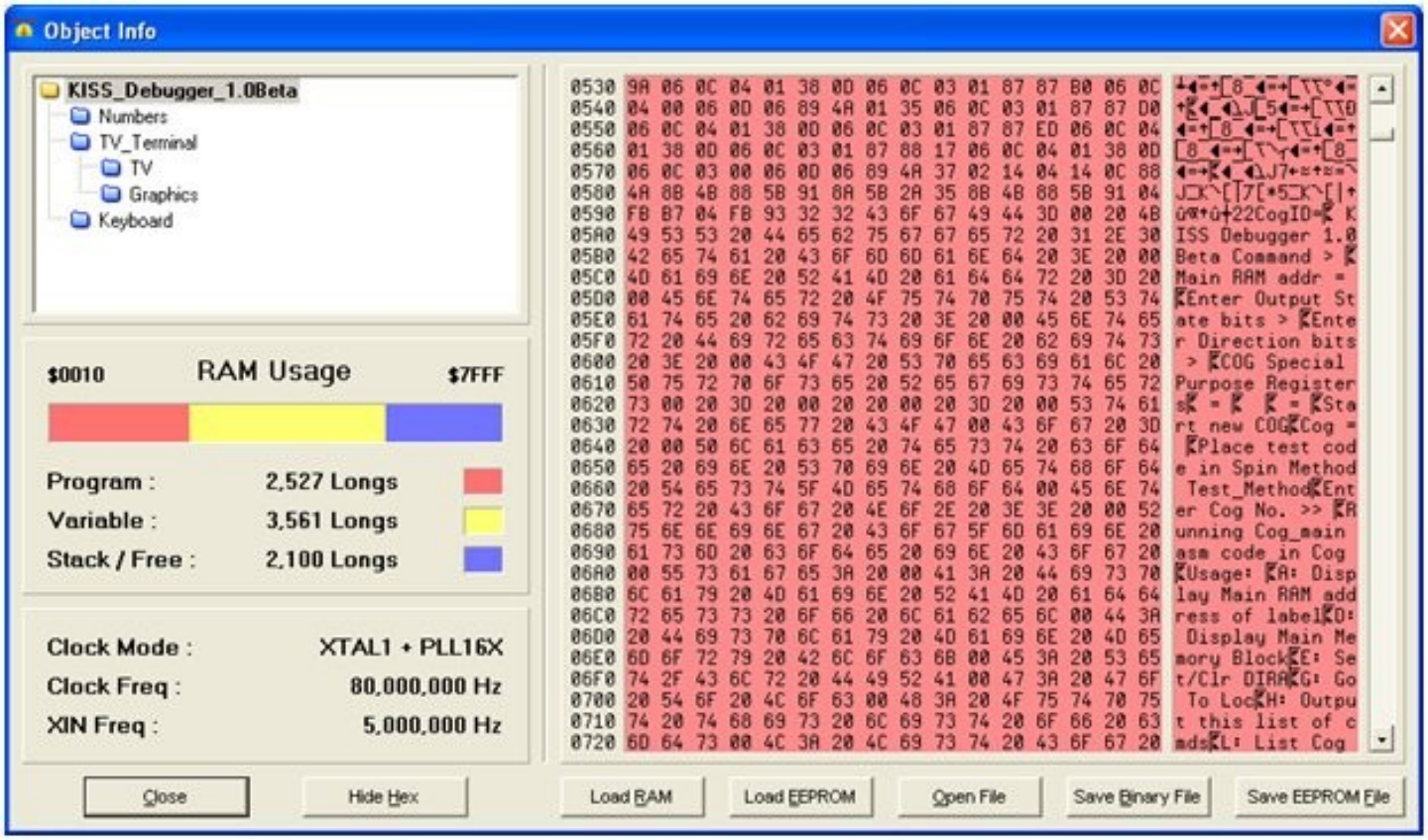


IMAGE 7
The Propeller Tool window for viewing detailed program information

How to use the Debugger

The Debugger can be used in two modes, either as the top level user interface wrapper or as a lower level user interface. When used as the top level user interface, the routine to be tested is be inserted into the Debugger "T" command and the variable name to be watched is inserted into the "A" command. You then compile all of the code using the Propeller Tool compiler/assembler and the resultant hex file is automatically downloaded into either the RAM or EEPROM depending upon whether the F10 or F11 key is used to initiate the compile. The Debugger automatically starts running and outputs:

```
"Cog ID = KISS Debugger 1.0 Beta? Command > "
```

You would then type "A" to obtain the absolute RAM address of the variable to be watched. The Debugger would respond with "Main RAM addr = xxxxxxxx". Next you type "T" to execute the code to be tested. Then you would use the "D" or "M" commands along with the address obtained from the "A" command to examine the results. While this is a simplistic example, it does show how you might employ the Debugger to test a routine. To try out this user routine, you do not have to generate a user interface but rather only insert the code to be tested into the Debugger "T" command structure and recompile. This clearly shows how the Debugger saves both time and frustration. A debugger always earns its keep.

The second way to use the Debugger is to include the Debugger as a lower level subset of your project code and invoke it by placing a call to it at the location in your program that you wish to debug. You would then compile and load the complete program using the F10 key which will download it to Propeller RAM. When the program encounters the call to the debugger during execution, the debugger starts a new Cog and outputs the debug command line to the screen. This allows you to use the debugger as a "BREAKPOINT" and examine whatever you wish in the system. The debugger call statement can be moved as needed to debug your program. If you are using most of MAIN RAM for your project, you can omit portions of Debugger code that you don't need, such as the Help command.

Using the debugger in one of the previously described ways, you can debug either a single instruction or a complete program in the real-time environment.

Let's take a look at an essential Debugger routine that you might also find useful in your projects. The User_Conv_Hex routine converts ASCII 0-9 & A-F to HEX $0-F:

```
PUB User_Conv_Hex                    'routine to convert ASCII to HEX
  {{ Convert char to hex digit }}
   Char_Buf &= $1f                   'mask off upper 3 bits
   If Char_Buf < $10
        Char_Buf += 9
        Char_Buf &= $f               'if less than $10 return
        return
   Char_Buf -= $10                   'subtract $10,mask with $F and return
   Char_Buf &= $f
```

There are seven executable SPIN statements and two non executable statements in this routine. This routine is a good example of the type of optimized code that can be written and tested using our debugger.

In future articles, we'll discuss fault-tolerant coding techniques, Dr. Gouge will share some insights on digital servo drivers based on his experience with servo electronics design and we will dig into decision making based on machine intelligence. We will make extensive use of our debugger as we refine complex Spin and Assembly

Language routines to incorporate these capabilities into our platform.

We hope you will choose to follow the articles in Robot Magazine and build a truly powerful and flexible platform for your robot project based on the Parallax Propeller chip.

Do not be intimidated if you do not understand everything right away. You will eventually. We will provide email technical support for free. If that is not enough to resolve an assembly error, we also provide an assembly repair service. You simply ship your board to our facility and we will repair it for a flat fee of $24.95 plus shipping and handling. If the board has burned out, we will contact you, as there may be an additional fee for a new board.

We invite you to check our website regularly at www.machineinteltech.com for updates and expansions of this project.

Next month we will tackle the memory expansion board. Until then, we will see you online.

Happy building!

James O. Gouge has a PhD in Computer Science .

Mark Allred has an undergraduate certificate in computer programming from Southern Polytechnic State University in Marietta, Georgia.

Our email address is support@machineinteltech.com. We welcome your comments and suggestions.

Links
Parallax, www.parallax.com
Machine Intelligence Technologies, www.machineinteltech.com