Adding Your Own Block Functions

There may be functions you need that are not included. Examples would be log, sine, tan, etc. Perhaps you have a device you want to add that runs off the SPI bus. The process of adding a block function takes a few steps but none are complicated.

We will proceed with an example that adds a break point function. The function takes an input and sends it to the output multiplied by a gain of 1. The output equals the input. If the input is greater than the positive breakpoint, the output will change to output = breakpoint + (input – breakpoint) * gain. A similar action occurs when the input drops below the negative breakpoint. Our system only allow five variables per block. So we have input, +breakpoint, -breakpoint, gain, and output variables.

Open Pico_block_compiler_i using the Arduino IDE. Then save it as Pico_block_compiler_j

We are going to add new block called break point and use bpt as the block name. Scroll down until you find the End Block Definitions comment. Define the new block function name in capital letters and use the next available block number 41. Next, increment BLK_MAX by one which is 42.

```
68  #define RLY 38
69  #define NOC 39
70  #define NCC 40
71  #define BPT 41
72
73  #define BLK_MAX 42
74  #define PMAX 50
75  #define VMAX 100
76  #define TIMING_PIN 22
77
78  //****End Block Definitions
79
```

Locate the getnames() definition. Right above it the printnames() function. We are going to copy the last if statement, paste it and change it to print “bpt” when the constant BPT is found.

```
385     if (b == ANG) Serial.print("and ");
386     if (b == ORG) Serial.print("ore ");
387     if (b == XOR) Serial.print("xor ");
388     if (b == RLY) Serial.print("rly ");
389     if (b == NOC) Serial.print("noc ");
390     if (b == NCC) Serial.print("ncc ");
391     if (b == BPT) Serial.print("bpt ");
392 }
393
394
395 int getnames(int n) {
396 volatile int j;
```

Find the print help command section. Add some help text for our new block.

```
505     Serial.println("var avg : number of adc sample to average per reading");
506     Serial.println("");
507     Serial.println("abs in out : out = |in|");
508     Serial.println("adc channel out  : read adc channel 0,1 at +-10V, 2 at 0 to 3.3V");
509     Serial.println("and in1 in2 out : out = 1 if in1>0 and in2>0 else out = 0");
510     Serial.println("bnz in offset :if in!=0 then PC=PC+offset else PC=PC+1");
511     Serial.println("bpt in bp+ bp- g out :out=in until bp then out = in*g");
512     Serial.println("brm in offset :if in<0 then PC=PC+offset else PC=PC+1");
513     Serial.println("brp in offset :if in>0 then PC=PC+offset else PC=PC+1");
514     Serial.println("brz in offset :if in==0 then PC=PC+offset else PC=PC+1");
```

Find the No Op command section. Copy and paste the No Op code. We are going to modify it.

```
650
651   //No Op
652   if ((cmd_buf[0] == 'n') && (cmd_buf[1] == 'o') && (cmd_buf[2] == 'p')) {
653    prog_array[prog_count][0] = NOP;
654    num_var[NOP] = 0;
655    return(getnames(0));
656   }
657
658   //Break Point Function
659   if ((cmd_buf[0] == 'b') && (cmd_buf[1] == 'p') && (cmd_buf[2] == 't')) {
660    prog_array[prog_count][0] = BPT;
661    num_var[BPT] = 5;
662    return(getnames(5));
663   }
664
```

Change the comment to Break Point Function. Change ‘n’ to ‘b’, ‘o’ to ‘p’ and ‘p’ to ‘t’. Change NOP to BPT. Change num_var[NOP] = 0; to num_var[BPT]=5;. Finally, getnames(0) to getnames(5). This is the command interpreter section. When it finds “bpt” it will enter the BPT constant into the program array. Next, it will look for five variables. If they are not defined, it will create them.

```
//No Op
if ((cmd_buf[0] == 'n') && (cmd_buf[1] == 'o') && (cmd_buf[2] == 'p')) {
 prog_array[prog_count][0] = NOP;
 num_var[NOP] = 0;
 return(getnames(0));
}

//Break Point Function
if ((cmd_buf[0] == 'b') && (cmd_buf[1] == 'p') && (cmd_buf[2] == 't')) {
 prog_array[prog_count][0] = BPT;
 num_var[BPT] = 5;
 return(getnames(5));
}
```

Scroll down until you find the run command section. It has all the case statements. Locate the LIM case and copy then paste it.

```
987       break;
988     case LIM:
989       limiter(prog_array[i][1],prog_array[i][2],prog_array[i][3],prog_array[i][4]);
990       i++;
991       break;
992     case LIM:
993       limiter(prog_array[i][1],prog_array[i][2],prog_array[i][3],prog_array[i][4]);
994       i++;
995       break;
996     case ZIN:
997       invz(prog_array[i][1],prog_array[i][2]);
```

Change case LIM to BPT. Change the calling function limiter to breakpt. Add one more variable prog_array[i][5] to the list.

```
988     case LIM:
989       limiter(prog_array[i][1],prog_array[i][2],prog_array[i][3],prog_array[i][4]);
990       i++;
991       break;
992     case BPT:
993       breakpt(prog_array[i][1],prog_array[i][2],prog_array[i][3],prog_array[i][4],prog_array[i][5]);
994       i++;
995       break;
996     case ZIN:
997       invz(prog_array[i][1],prog_array[i][2]);
```

When the run command is executed, the program reads the block constant from prog_array[i][0]. The switch statement then calls the block function and passes an array index of the variables to the block function. As you guessed, i is the program line number.

As before, we will copy, paste and modify code to add the breakpoint function. Find the limiter routine. To make it a little easier, copy this code and paste it below the limiter function.

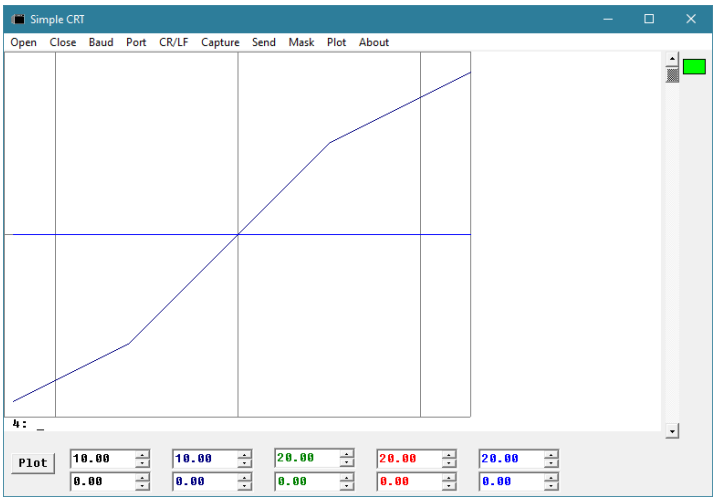
```
void breakpt(int a1, int bp, int bm, int g, int s1){
 val_array[s1][1] = val_array[a1][1];
 if (val_array[a1][1] > val_array[bp][1]) val_array[s1][1] = val_array[bp][1] + (val_array[a1][1]-val_array[bp][1])*val_array[g][1];
 if (val_array[a1][1] < val_array[bm][1]) val_array[s1][1] = val_array[bm][1] + (val_array[a1][1]-val_array[bm][1])*val_array[g][1];
}
```

```
1378 void limiter(int al, int pl, int ml, int sl){
1379  val_array[sl][1] = val_array[al][1];
1380  if (val_array[al][1] > val_array[pl][1]) val_array[sl][1] = val_array[pl][1];
1381  if (val_array[al][1] < val_array[ml][1]) val_array[sl][1] = val_array[ml][1];
1382 }
1383
1384 void breakpt(int al, int bp, int bm, int g, int sl){
1385  val_array[sl][1] = val_array[al][1];
1386  if (val_array[al][1] > val_array[bp][1]) val_array[sl][1] = val_array[bp][1] + (val_array[al][1]-val_array[bp][1])*val_array[g][1];
1387  if (val_array[al][1] < val_array[bm][1]) val_array[sl][1] = val_array[bm][1] + (val_array[al][1]-val_array[bm][1])*val_array[g][1];
1388 }
1389
1390 void schmidt(int al, int tl, int sl){
1391  if (val_array[al][1] > val_array[tl][1]) val_array[sl][1]= -1.0;
1392  if (val_array[al][1] < -val_array[tl][1]) val_array[sl][1]= 1.0;
1393 }
```

The array indexes passed to the breakpt function are used to find the variables in the val_array[n][1]. The second index of [n][1] is used for all immediate output changes. Some functions such as integration save the output at index [n][0] and it is updated at the end of the program to reside at index [n][1]. Locations [s1][2] and [s1][3] for the output variable can be used as non-volatile local storage. Don't use these locations on input variables since they are outputs of other block functions.

Save the changes. If all is well with your entry, you should be able to compile and upload the program to your Pico. Here is a simple program to ramp an input from -12.8 to 12.8. The breakpoints are set at +5 and -6 with a gain of 0.5.

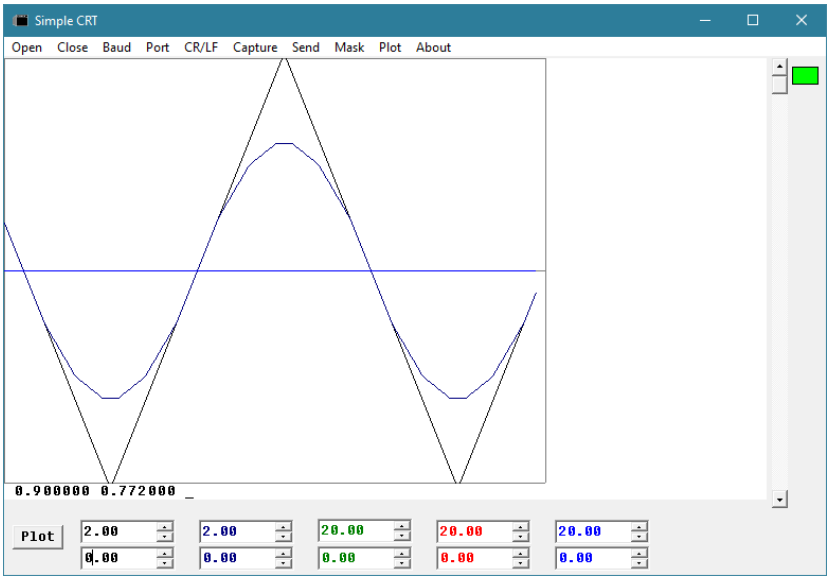
```
clr
sum a dt a
bpt a bp bm g out
prt a out
end
set a -12.8
set dt 0.1
set max -25.6
set bp 5
set bm -6
set g 0.5
```



An x-y plot the output shows the breakpoint function is working as planned.

Breakpoint functions can be cascaded. Just remember the gains are multiplied. This example generates a triangle wave, passes it through two breakpoint blocks and then clips the output with a limiter.

```
clr
smt in th out
eul out g lim in
bpt in p1 n1 g1 s1
bpt s1 p2 n2 g2 s2
lim s2 p3 n3 sin
prt in sin
end
set th 2
set out 1
set dt .01
set max 100
set g 5
set p1 .5
set n1 -.5
set g1 .68
set p2 1
set n2 -1
set g2 .47
set p3 1.2
set n3 -1.2
```



A piecewise sine approximation is what you get. The gain can also be negative. So, the new block could be uses to simulated a device with negative resistance.

Running Your Block Program at Power Up

It is possible to make a stand alone setup that requires no program loading through the serial interface. First you will need to write and test your program. Next, it will need to be modified so it can be stored in the Pico code. Start by adding a ***#define BLOCKPROG "*** to the clr statement. End it and each following line with a **\r**. Add one more line ***%"***. Save the modified text. It will get inserted into the C++ code. Below is an example of modified block code.

```
//embedded block code. runs on boot if D0 is grounded
#define BLOCKPROG "clr\r\
sum n .01 n\r\
sub n x d\r\
brm d 2\r\
out L 25\r\
sub n 1 d\r\
brm d 4\r\
out H 25\r\
rst 0 n\r\
rst t x\r\
sub t 1 d\r\
brm d 2\r\
rst 0 t\r\
end\r\
set 1 1\r\
set 2 2\r\
set 4 4\r\
set 25 25\r\
set max 2\r\
set dt 0.0001\r\
set .01 .01\r\
set H 1\r\
set L 0\r\
run\r\
%"
```

Open Pico_block_compiler_j in the Arduino IDE. Located the End Block Definitions.

```
70 #define NCC 40
71 #define BPT 41
72
73 #define BLK_MAX 42
74 #define PMAX 50
75 #define VMAX 100
76 #define TIMING_PIN 22
77
78
79 //****End Block Definitions
80
81
82 //****System Variables
```

Cut and paste your modified block program just above the End Block Definitions.

```
92  end\r\
93  set 1 1\r\
94  set 2 2\r\
95  set 4 4\r\
96  set 25 25\r\
97  set max 2\r\
98  set dt 0.0001\r\
99  set .01 .01\r\
100 set H 1\r\
101 set L 0\r\
102 run\r\
103 %"
104
105 //****End Block Definitions
106
```

Locate the Serial character read section. We are going to modify these two lines.

```
260   line_count = 0;
261   Serial.print(prog_count);
262   Serial.print(": ");
263
264   while (c != 13) {
265     while (Serial.available() == 0);
266     c = Serial.read();
267     if (c != 10) {
268       Serial.write(c);
269       if (c==8){              //Back Space
270         line_count--;
271         if (line_count < 0) line_count = 0;
272       } else {
273        line_buf[line_count++] = c;
274        if (line_count > 79) line_count = 79;
275       }
276     }
277   }
```

Copy this code then replace the two lines with it.

```
if ((digitalRead(0)==1)||(sst!=0)){
  while (Serial.available() == 0);
  c = Serial.read();
 } else
 {
  if (sst ==0)
   c = BLOCKPROG[ist];
  if (c != '%') ist++; else {
    sst++;
    c ='\0';
  }
 }
```

```
261   Serial.print(prog_count);
262   Serial.print(": ");
263
264   while (c != 13) {
265   if ((digitalRead(0)==1)||(sst!=0)){
266     while (Serial.available() == 0);
267     c = Serial.read();
268   } else
269   {
270     if (sst ==0)
271      c = BLOCKPROG[ist];
272     if (c != '%') ist++; else {
273       sst++;
274       c = '\0';
275     }
276   }
277
278     if (c != 10) {
279       Serial.write(c);
280       if (c==8){              //Back Space
```

Compile and upload the program to your Pico. If you used the example block code, it will make the LED flash from dim to bright. Place a jumper wire from D0 to GND, then plug in the USB cable. This will make the Pico boot to the imbedded program.

Connect to the terminal program and hit <esc>. The boot program will stop and you can list it. The program will only load on boot. If you clear it or change lines, ground D0 and repower the Pico to get it back.