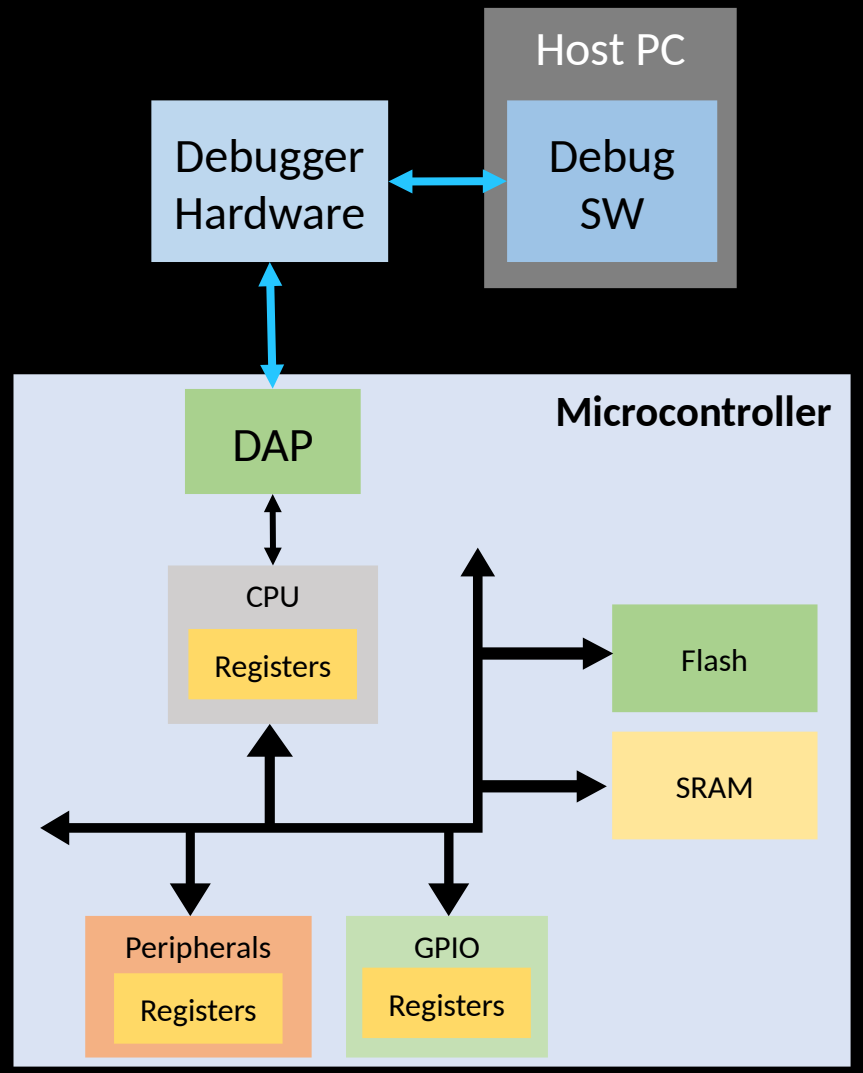
Debugging a Microcontroller Program: Part 1

Embedded Software Essentials

C2 M3 V7

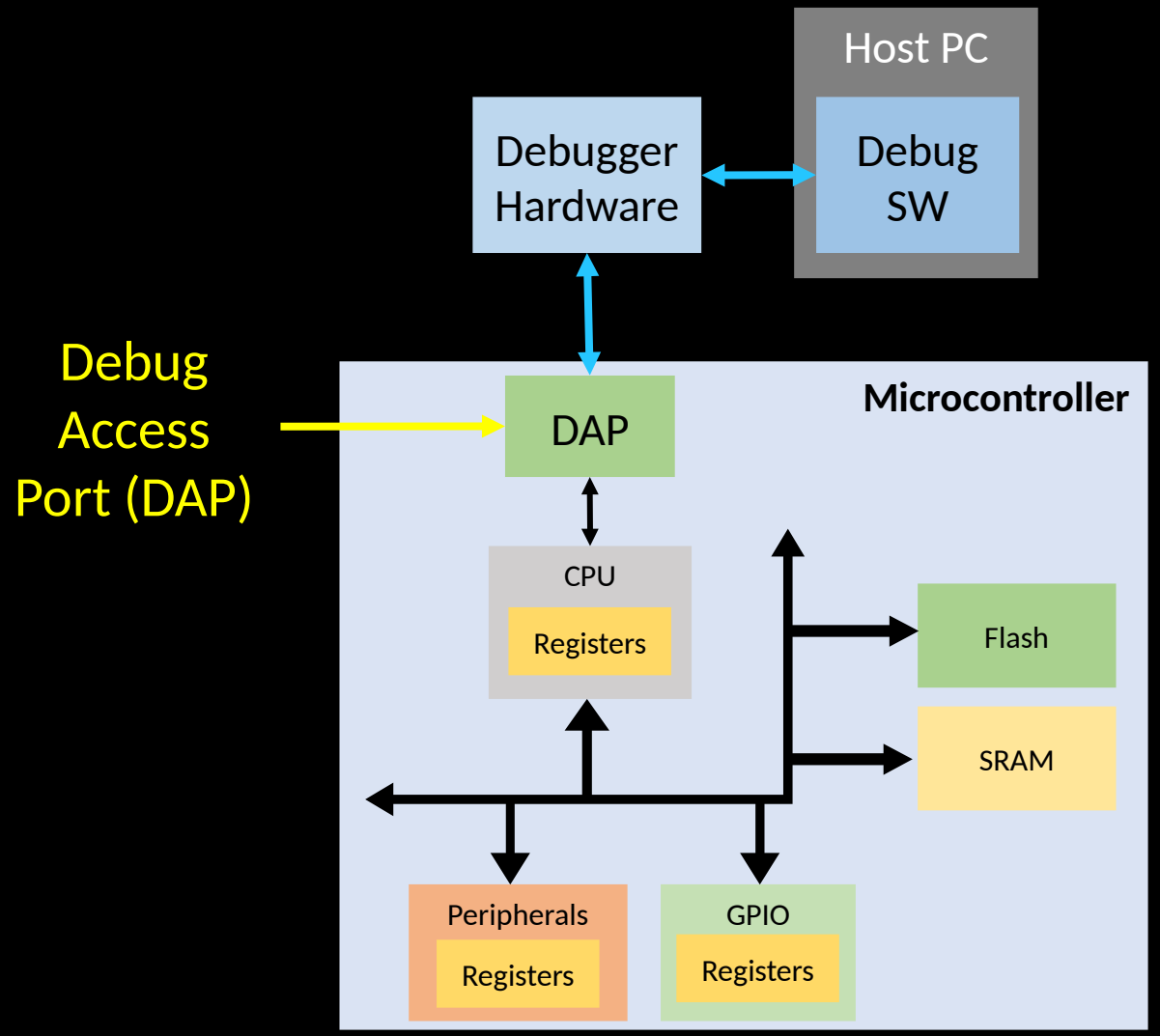
Testing Software [S1a]

- Debugger: Software application that connects to embedded system's target application



Testing Software [S1b]

- Debugger: Software application that connects to embedded system's target application

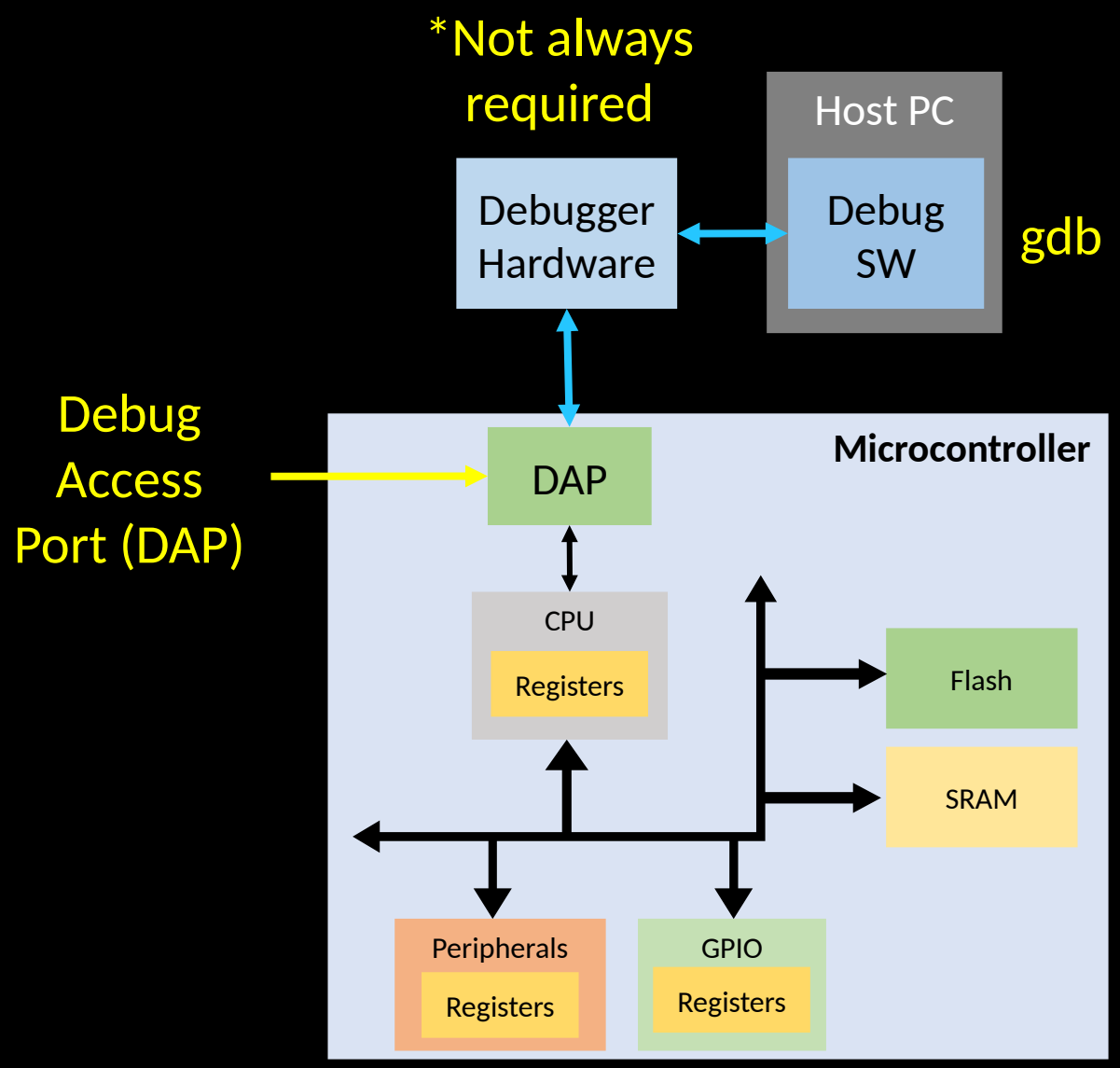


- Debuggers allow users to
  - Observe Behavior
  - Control Behavior

Testing Software [S1c]

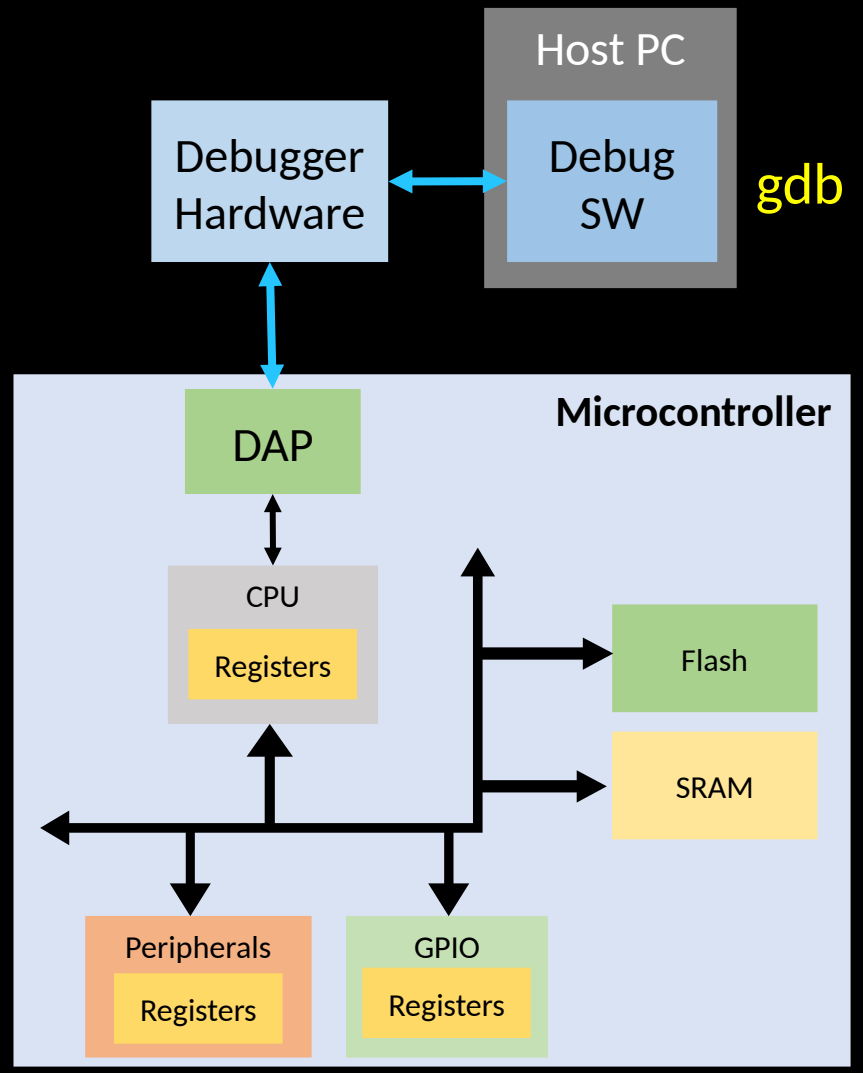
- Debugger: Software application that connects to embedded system's target application
  - E,g: GNU Debugger (gdb)

- Debuggers allow users to
  - Observe Behavior
  - Control Behavior



Debuggers [S2a]

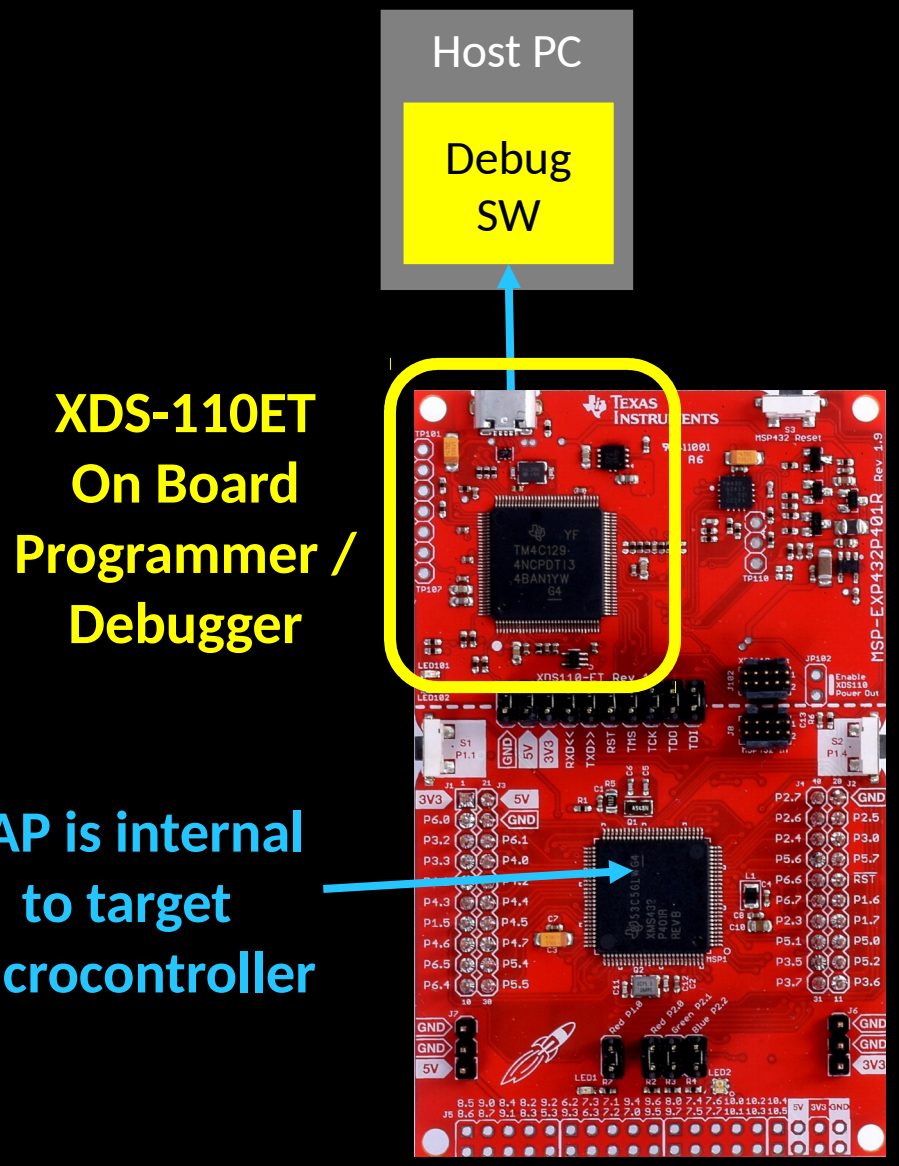
- Main Tasks:
  - Connect to development board
  - Program an application
  - Debug an application



Debuggers [S2a]

- Main Tasks:
  - Connect to development board
  - Program an application
  - Debug an application

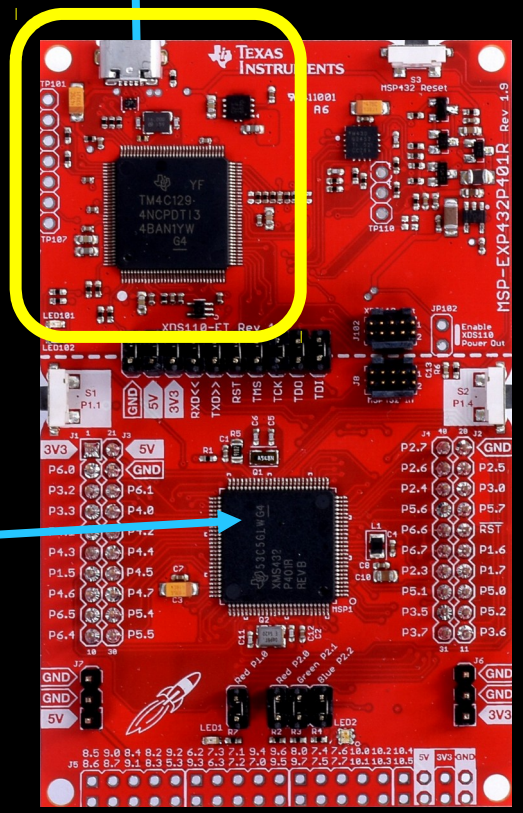
- Requires software application running on a host machine



Debuggers [S2c]

- Main Tasks:
  - Connect to development board
  - Program an application
  - Debug an application

- Requires software application running on a host machine
  - Code Composer Studio (CCS)
    - Texas Instruments IDE with Debugger

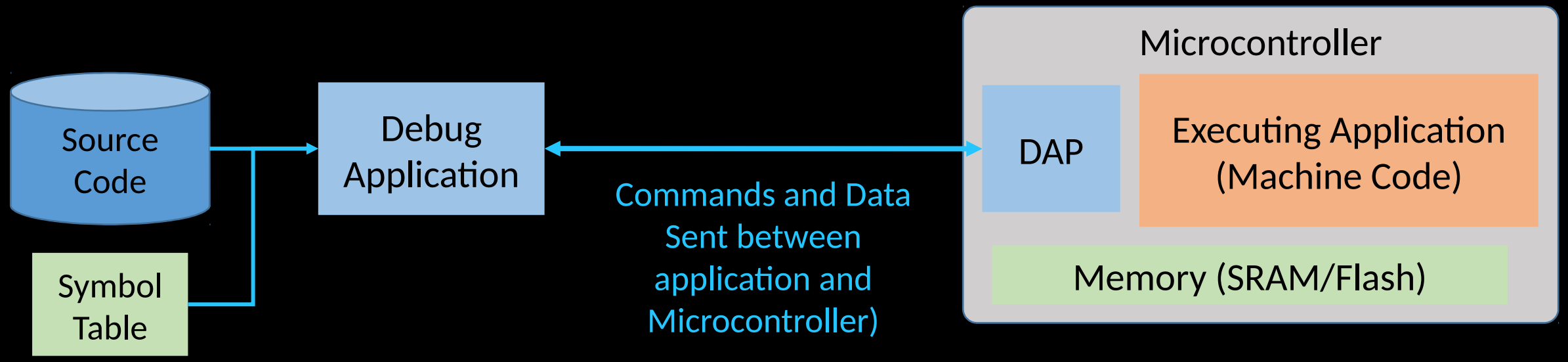


Debug Symbols [S3]

- Must enable debug symbols at compilation
  - -g flag

```
$ arm-none-eabi-gcc –g main.c –o main.o
```

- Provides mechanism for debugger to interpret executing code and map to original source code



Debug Access Port [S4a]

- Built in debug port called Debug Access Port (DAP)
- Supports multiple debugging interfaces
  - Joint Test Action Group (JTAG)
  - Serial Wire Debug (SWD)

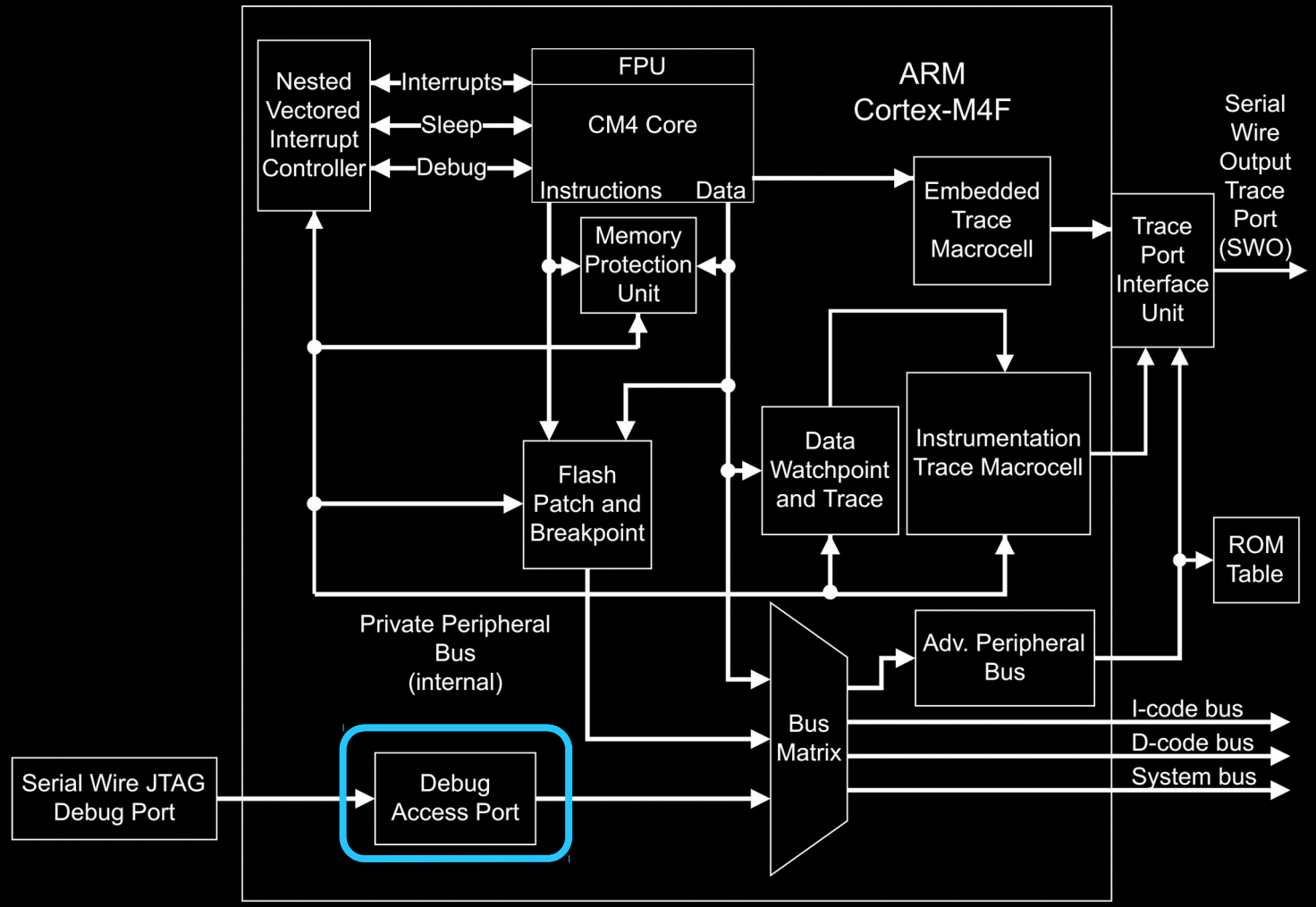


Figure 1-1. CPU Block Diagram

Debug Access Port [S4a]

- Built in debug port called Debug Access Port (DAP)
- Supports multiple debugging interfaces
  - Joint Test Action Group (JTAG)
  - Serial Wire Debug (SWD)

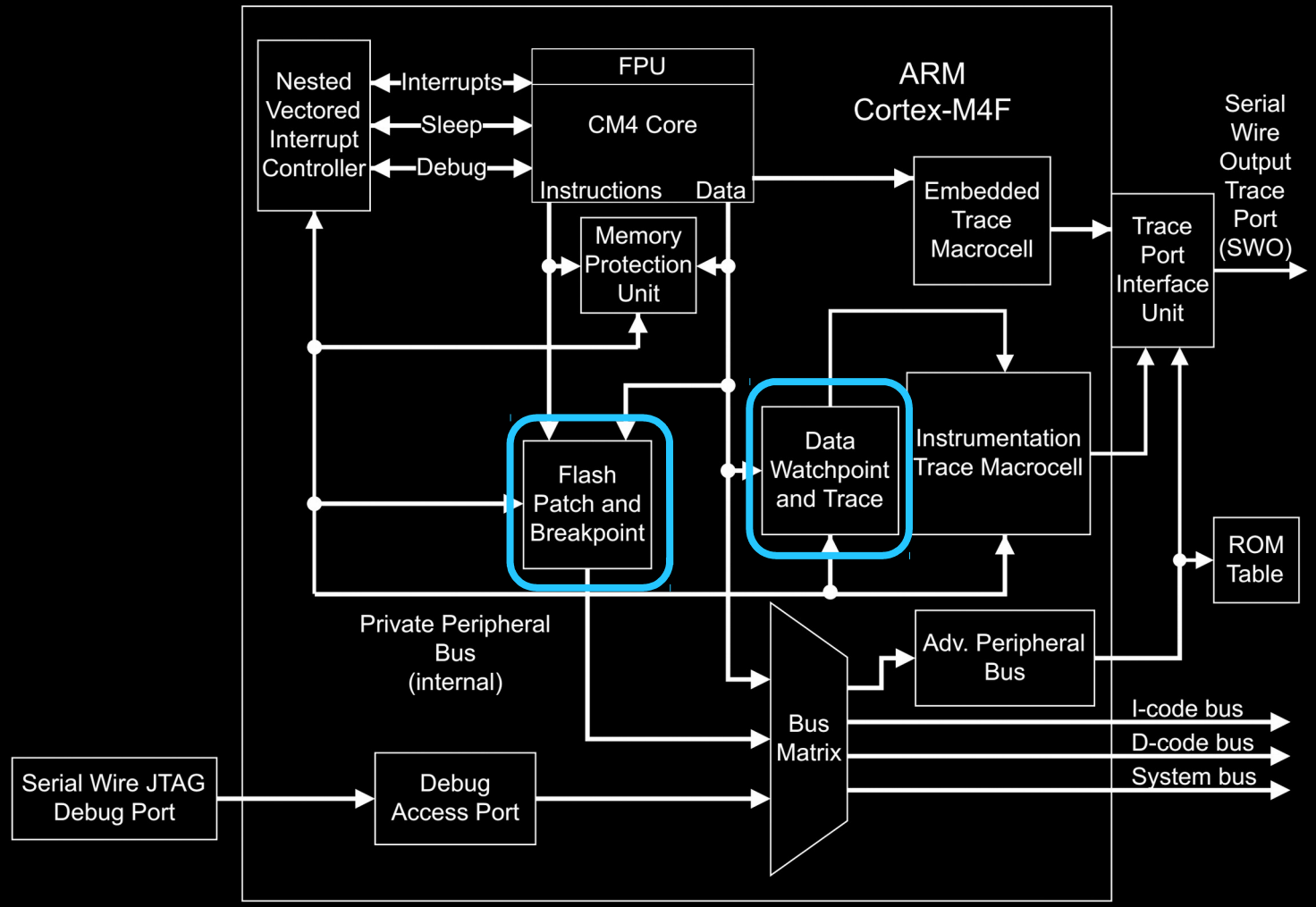


Figure 1-1. CPU Block Diagram

Breakpoints [S5a]

- Mechanism to stop or pause the current execution of a program
  - Program must be un-paused or restarted a breakpoint occurs

blink.c

```
48    /* Stop watchdog timer */
49    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;
50
51    /* Configure P1.0 as output */
52    P1->DIR |= BIT0;
53
54    /* Initialize Memory for a Memory Dump Debugging E
55    for( i = 0; i < ARRAY_MAX; i++)
56    {
57        array1[i] = i;
58    }
59    my_memset(array2, ARRAY_MAX, 0xff);
60
61    /* Create a NOP instruction for a breakpoint */
62    __NOP();
```

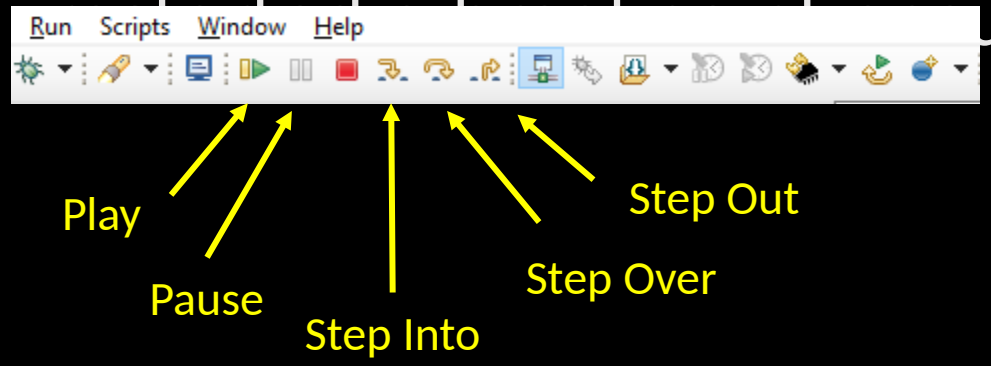
(x)= Variables | Expressions | Registers | Breakpoints

Identity	Name
☐ 0x2000fff8 (__stack + 0x1F8) [H/W BP - disabled]	Watchpoint
☑ main.c, line 52 (main + 0xE) [S/W BP]	Breakpoint
☐ main.c, line 59 (CL2) [S/W BP - disabled]	Breakpoint
☑ main.c, line 67 (CL2 + 0x18) [S/W BP]	Breakpoint

- Limited number of breakpoints allowed

Breakpoints [S5b]

- Mechanism to stop or pause the current execution of a program
  - Program must be un-paused or urs



- Limited number of breakpoints allowed

Breakpoint Indicator

blink.c

```
48  /* Stop watchdog timer */
49  WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;
50
51  /* Configure P1.0 as output */
52  P1->DIR |= BIT0;
53
54  /* Initialize Memory for a Memory Dump Debugging Ex
55  for( i = 0; i < ARRAY_MAX; i++)
56  {
57      array1[i] = i;
58  }
59  my_memset(array2, ARRAY_MAX, 0xff);
60
61  /* Create a NOP instruction for a breakpoint */
62  __NOP();
```

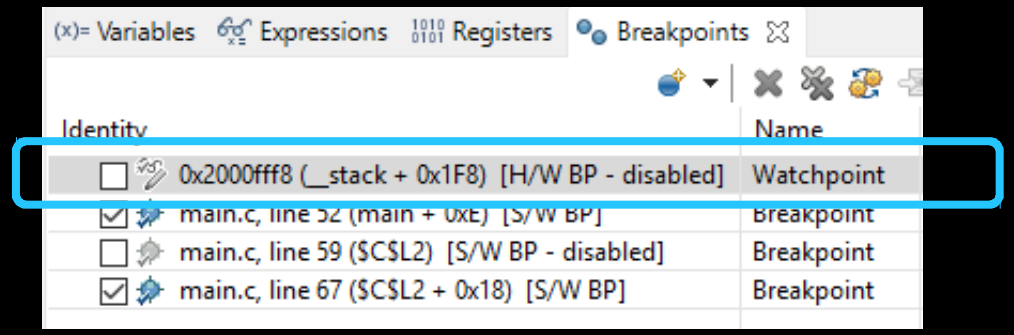
(x)= Variables | Expressions | Registers | Breakpoints

Identity	Name
0x2000fff8 (__stack + 0x1F8) [H/W BP - disabled]	Watchpoint
main.c, line 52 (main + 0xE) [S/W BP]	Breakpoint
main.c, line 59 (CL2) [S/W BP - disabled]	Breakpoint
main.c, line 67 (CL2 + 0x18) [S/W BP]	Breakpoint

Watch Points [S6a]

- Mechanism to stop or pause the current execution of a program when a variable or expression changes

- Can happen at any point in the program

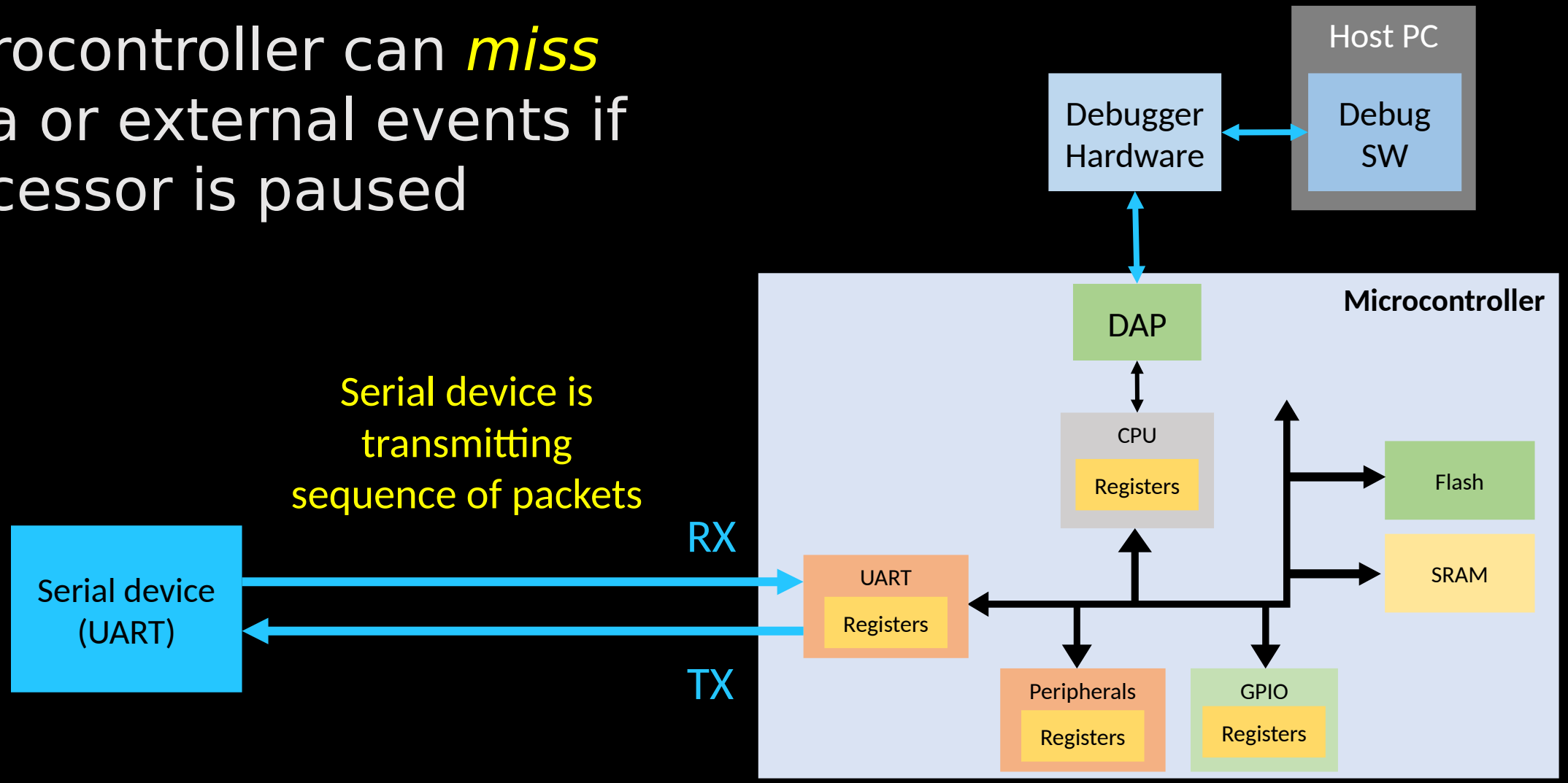


Example:
Set watch point on variable i. Anytime i changes, break. This should occur 16 times in this code excerpt

```
uint32_t i = 0;
for (i = 0; i < 16; i++)
{
    /* Other Code Here*/
}
```

Breakpoints & Watchpoints [S7a]

- Microcontroller can *miss* data or external events if processor is paused



Variable Scope [S8a]

- Variables view allows you to see value while in current scope
- Must be paused to read value
- Can inject any value during a pause for testing

Before Entry to Main

```
44 void main(void)
45 {
46     volatile uint32_t i = 0;
47
48     /* Stop watchdog timer */
49     WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;
```

```
44 void main(void)
45 {
46     volatile uint32_t i = 0;
47
48     /* Stop watchdog timer */
49     WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;
```

Before Initialization

(x)= Variables | Expressions | Registers | Breakpoints

Name	Type	Value
(x)= i	unknown	Error: Memory map prevented reading 0x20010000

After Initialization

(x)= Variables | Expressions | Registers | Breakpoints

Name	Type	Value
(x)= i	unsigned int	0

Expressions [S9a]

- Expressions view allows you to see any global variable, register or general expression
- Can write c-expressions in here and they are evaluated at runtime
- Must be paused to read values

Breakpoint after array1 initialized

(x)= Variables | Expressions | Registers | Breakpoints

Expression	Type	Value	Address
array1	unsigned char[16]	0x20000428 (Hex)	0x20000428
(x)= [0]	unsigned char	0x00 (Hex)	0x20000428
(x)= [1]	unsigned char	0x01 (Hex)	0x20000429
(x)= [2]	unsigned char	0x02 (Hex)	0x2000042A
(x)= [3]	unsigned char	0x03 (Hex)	0x2000042B
(x)= [4]	unsigned char	0x04 (Hex)	0x2000042C
(x)= [5]	unsigned char	0x05 (Hex)	0x2000042D
(x)= [6]	unsigned char	0x06 (Hex)	0x2000042E
(x)= [7]	unsigned char	0x07 (Hex)	0x2000042F
(x)= [8]	unsigned char	0x08 (Hex)	0x20000430
(x)= [9]	unsigned char	0x09 (Hex)	0x20000431
(x)= [10]	unsigned char	0x0A (Hex)	0x20000432
(x)= [11]	unsigned char	0x0B (Hex)	0x20000433
(x)= [12]	unsigned char	0x0C (Hex)	0x20000434
(x)= [13]	unsigned char	0x0D (Hex)	0x20000435
(x)= [14]	unsigned char	0x0E (Hex)	0x20000436
(x)= [15]	unsigned char	0x0F (Hex)	0x20000437
array2	unsigned char[16]	0x20000438 (Hex)	0x20000438
(x)= i	unsigned int	0x00000010 (Hex)	0x2000FFF8
Add new expressi			

```
54  /* Initialize Memory for a Memory Dump Debugging Example */
55  for( i = 0; i < ARRAY_MAX; i++)
56  {
57      array1[i] = i;
58  }
59  my_memset(array2, ARRAY_MAX, 0xff);
```

Registers [S10a]

- Register view allows you to view
  - Core CPU Registers
    - General Purpose
    - Special Purpose
  - Peripheral Registers

- Must be paused to read values

- Can inject any value into

x)= Variables | Expressions | Registers | Breakpoints

Name	Value	Description
Core Registers		Core Registers
PC	0x00000AD6	Program Counter [Core]
SP	0x2000FFF8	General Purpose Register 13 - Stack Pointer [Core]
LR	0x00000D6B	General Purpose Register 14 - Link Register [Core]
xPSR	0x61000000	Stores the status of interrupt enables and critical pr...
R0	0x00000010	General Purpose Register 0 [Core]
R1	0x0000000F	General Purpose Register 1 [Core]
R2	0x20000428	General Purpose Register 2 [Core]
R3	0xFFFFFFF1	General Purpose Register 3 [Core]
R4	0x00000000	General Purpose Register 4 [Core]
R5	0x00000000	General Purpose Register 5 [Core]
R6	0x00000000	General Purpose Register 6 [Core]
R7	0x20010000	General Purpose Register 7 [Core]

CS		
PA		
P1		
P1IV	0x0000	Port 1 Interrupt Vector Register [Memory Mapped]
P1IN	0xF7	Port 1 Input [Memory Mapped]
P1OUT	0xFF	Port 1 Output [Memory Mapped]
P1DIR	0x01	Port 1 Direction [Memory Mapped]
P1REN	0x00	Port 1 Resistor Enable [Memory Mapped]
P1DS	0x00	Port 1 Drive Strength [Memory Mapped]
P1SEL0	0x00	Port 1 Select 0 [Memory Mapped]
P1SEL1	0x00	Port 1 Select 1 [Memory Mapped]
P1SELC	0x00	Port 1 Complement Select [Memory Mapped]
P1IES	0xAB	Port 1 Interrupt Edge Select [Memory Mapped]
P1IE	0x00	Port 1 Interrupt Enable [Memory Mapped]
P1IFG	0xF8	Port 1 Interrupt Flag [Memory Mapped]

Memory Browser [S11a]

- Memory Browser allows you to view and alter memory directly at different accessible addresses
- Can format memory view many ways
- Must be paused to read values

Expression	Type	Value	Address
array1	unsigned char[16]	0x20000428 (Hex)	0x20000428
array2	unsigned char[16]	0x20000438 (Hex)	0x20000438

array1 & array2 addresses

Memory Browser

array1

0x20000428 - array1 <Memory Rendering 1>

8-Bit Hex - C Style

```
0x20000428  array1
0x20000428  0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07
0x20000430  0x08 0x09 0x0A 0x0B 0x0C 0x0D 0x0E 0x0F
0x20000438  array2
0x20000438  0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x20000440  0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x20000448  0x12 0xD3 0x08 0xC9 0x03 0x80 0x1B 0x0C
0x20000450  0x43 0x80 0x00 0x1D 0x12 0x1F 0xF8 0xD2
0x20000458  0x0A 0xE0 0x08 0xC9 0x03 0x70 0x1B 0x0A
0x20000460  0x43 0x70 0x1B 0x0A 0x83 0x70 0x1B 0x0A
```

Stack Trace [S12a]

- Allows you to determine the function call hierarchy by looking at the stack
  - Stack provides history of call frames
- Must be paused to read current call stack
- Very helpful for multi-threaded programs

Main calls blink_led_forever()

```
74 void blink_led_forever()
75 {
76     uint32_t i;
77
78     /* Code to show LED blinking on and off *
79     while(1)
80     {
81         P1->OUT ^= BIT0;                    /
82         for( i = LENGTH1; i > 0; i--);      /
83     }
84 }
```

Main is called by _c_int00()