

SOC Design

Week 13: Performance Profiling and Debugging



Fenerbahçe University



Professor & TAs

Prof: Dr. Vecdi Emre Levent

Office: 311

Email: emre.levent@fbu.edu.tr

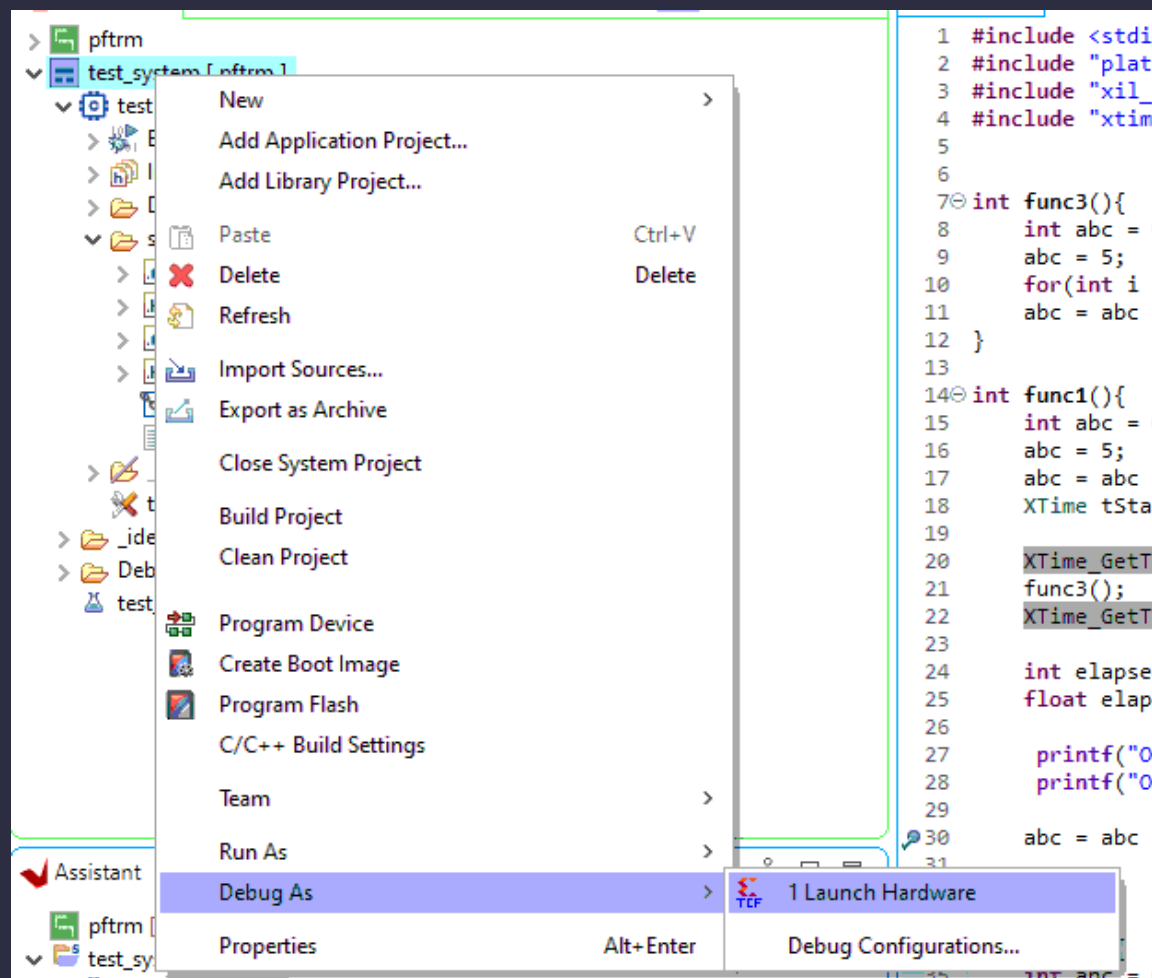
TA: Arş. Gör. Ezgi Çakmak

Office: 311

Email: ezgi.cakmak@fbu.edu.tr

Performance Profiling and Debugging

Debug başlangıcı



The screenshot shows an IDE interface. On the left, a project tree is visible with a context menu open over the 'test_system' project. The menu items include 'New', 'Add Application Project...', 'Add Library Project...', 'Paste', 'Delete', 'Refresh', 'Import Sources...', 'Export as Archive', 'Close System Project', 'Build Project', 'Clean Project', 'Program Device', 'Create Boot Image', 'Program Flash', 'C/C++ Build Settings', 'Team', 'Run As', 'Debug As', and 'Properties'. The 'Debug As' option is highlighted, and a sub-menu is open showing '1 Launch Hardware' and 'Debug Configurations...'. On the right, a code editor displays C code with performance profiling functions:

```
1 #include <stdi
2 #include "plat
3 #include "xil_
4 #include "xtim
5
6
7 int func3(){
8     int abc =
9     abc = 5;
10    for(int i
11    abc = abc
12 }
13
14 int func1(){
15     int abc =
16     abc = 5;
17     abc = abc
18     XTime tSta
19
20     XTime_GetT
21     func3();
22     XTime_GetT
23
24     int elapse
25     float elap
26
27     printf("0
28     printf("0
29
30     abc = abc
31
```

Performance Profiling and Debugging

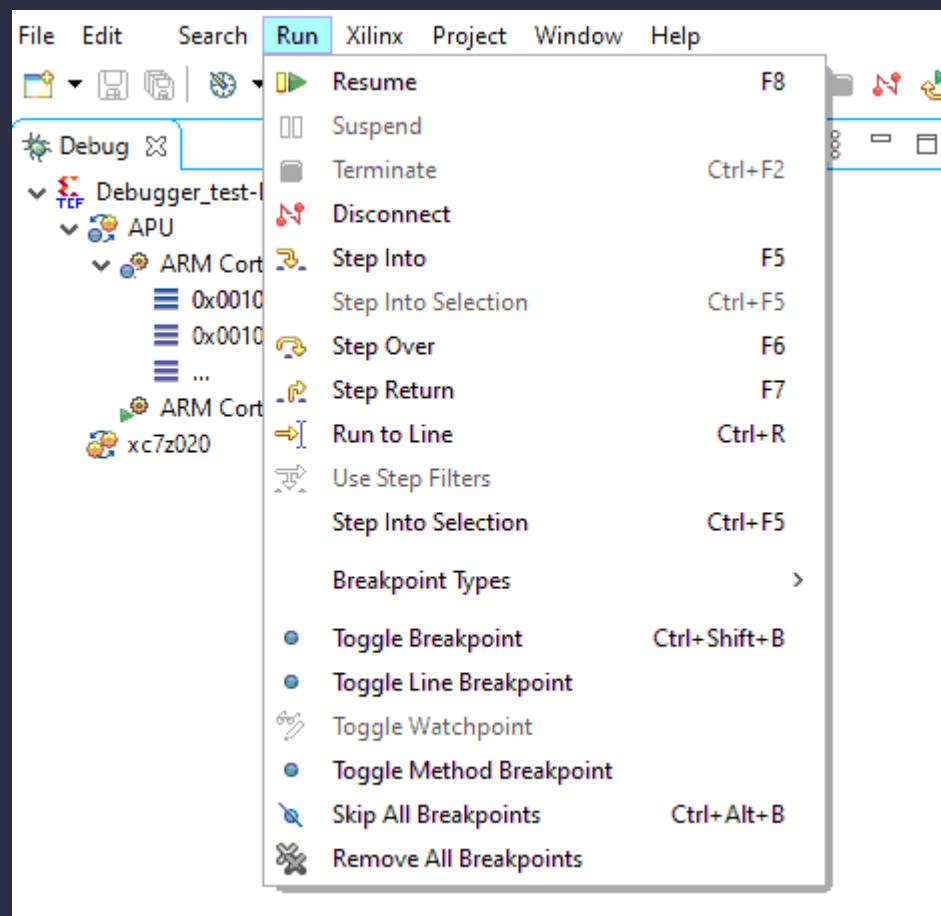
Eğer herşey doğru ise, main'in altında bulunan ilk satırdaki bulunan fonksiyonun üzerinde yeşil bir arka plan oluşacaktır.

Bu yeşil arka plan o anda beklenen satırı göstermektedir.

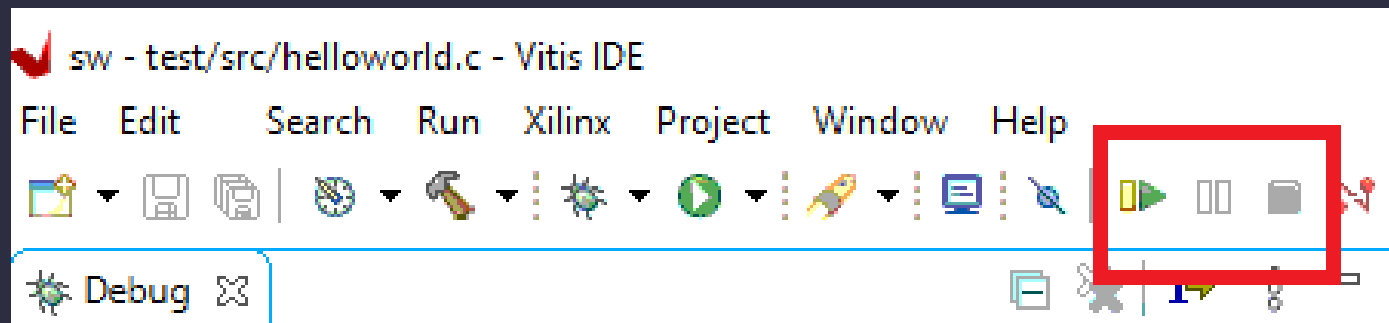
```
45
46 int main()
47 {
48     init_platform();
49
50
51     while(1){
52         func1();
53         func2();
54     }
55
56     print("Hello World\n\r");
57     print("Successfully ran Hello World application");
58
59
60     cleanup_platform();
61     return 0;
62 }
```

Performance Profiling and Debugging

Bu aşamada Run'un altındaki elemanlar debugging için kullanılabilir.

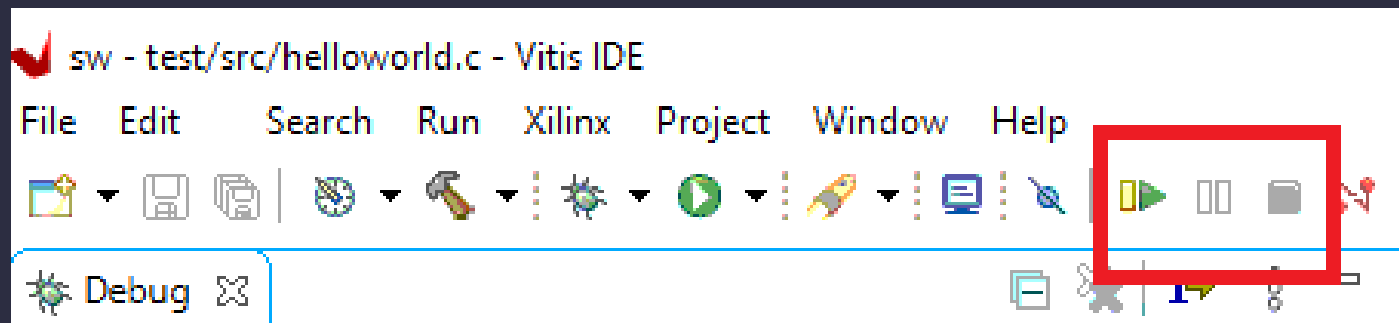


Performance Profiling and Debugging



Debugging başlatıldığında kırmızı kare içerisinde Resume, Suspend ve Terminate isimli butonlar görülecektir.

Performance Profiling and Debugging

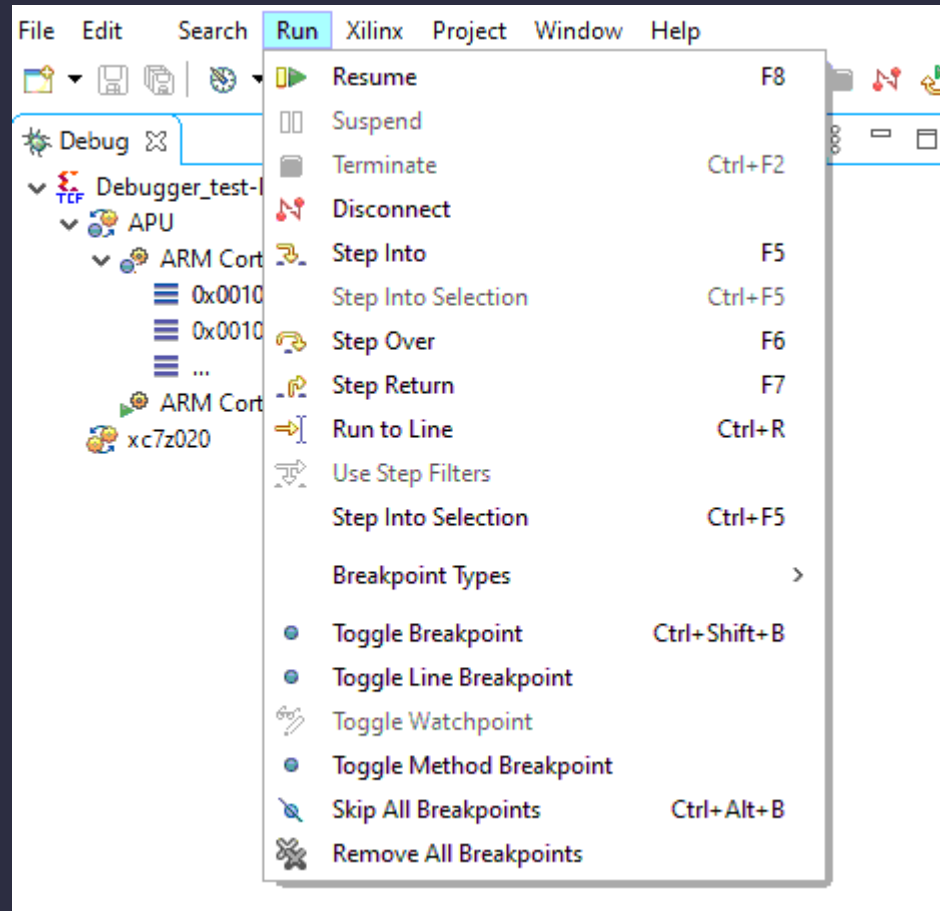


Resume, kodun yürütülmesini sağlar

Suspend, CPU o anki çalıştırdığı işi dondurur

Terminate, debugging durur

Performance Profiling and Debugging



Run butonunun alındaki elemanlar ile debugging yapılması:

Resume (F8)

Disconnect

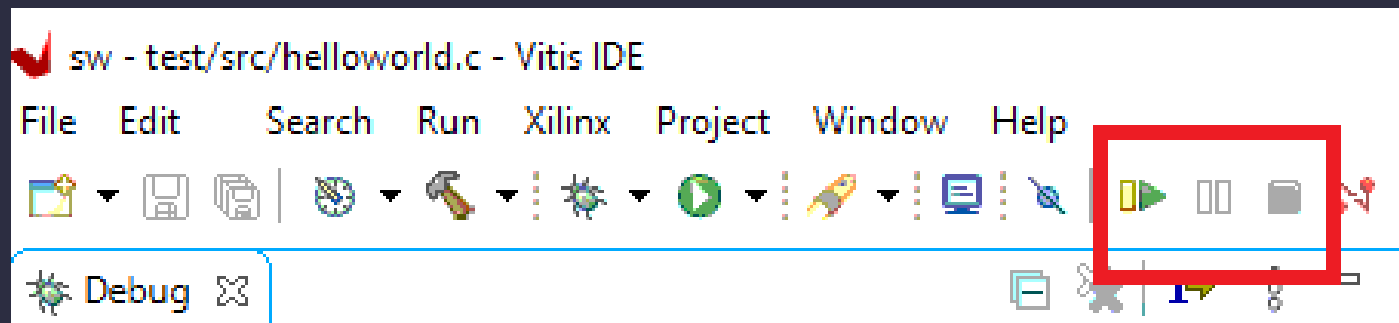
Step Into (F5) : Debug yapılan satır bir fonksiyonun üzerinde iken fonksiyonun içine dallanır. Fonksiyon değilse alt satırdan devam eder.

Step Over (F6): Doğrudan alt satırdan devam eder

Step Return (F7): Kodu devam ettirir

Return to Line (Ctrl + R): Aynı satıra ulaşılan kadar kod devam eder

Performance Profiling and Debugging



Resume, kodun yürütülmesini sağlar

Suspend, CPU o anki çalıştırdığı işi dondurur

Terminate, debugging durur

Performance Profiling and Debugging

```
23
24     int elapsedCycles = 2*(tEnd - tStart);
25     float elapsedUs = 1.0 * (tEnd - tStart) / (COUNTS_PER_SECOND/1000000);
26
27     printf("Output took %d clock cycles.\n", elapsedCycles);
28     printf("Output took %.2f us.\n", elapsedUs);
29
30     abc = abc * 10;
31
32 }
33
34 int func2(){
35     int abc = 0;
36     abc = 5;
37     abc = abc * 10;
38     abc = abc * 10;
39     abc = abc * 10;
40     abc = abc * 10;
41 }
```

Breakpoint, debugging esnasında bir satıra ulaşıldığı esnada durmasını sağlar.

Örneğin 27. satıra bir breakpoint konmuştur. Yürüt dendiğinde, o satırda kod duracaktır.

```
26
27     printf("Output took %d clock cycles.\n", elapsedCycles);
28     printf("Output took %.2f us.\n", elapsedUs);
```

Performance Profiling and Debugging

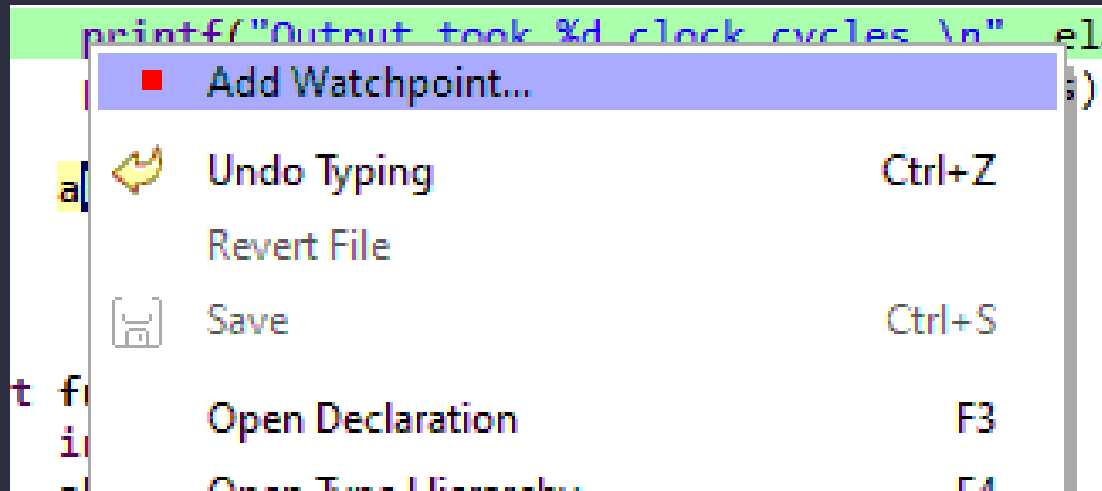
```
abc = abc * 10;
```

Name	Type	Value
(x)= abc	int	50

50
Hex: 00000032, Dec: 50, Oct: 062
Bin: 0000,0000,0000,0000,0000,0000,0011,0010
Size: 4 bytes, Type: int
Address: 0x11405c

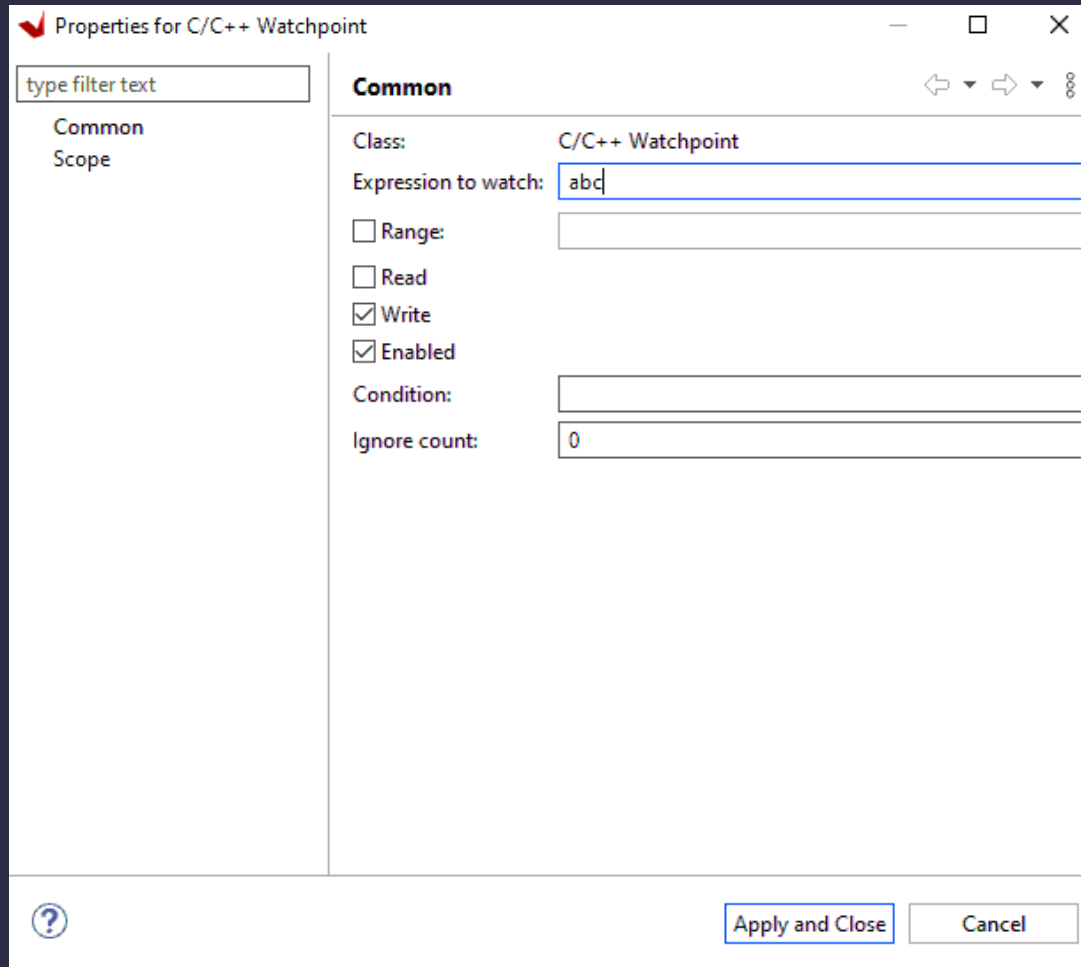
Debugging yapılırken, bir değişkenin üzerine mouse ile gelindiğinde, değişkenin o anki değeri görülebilmektedir.

Performance Profiling and Debugging



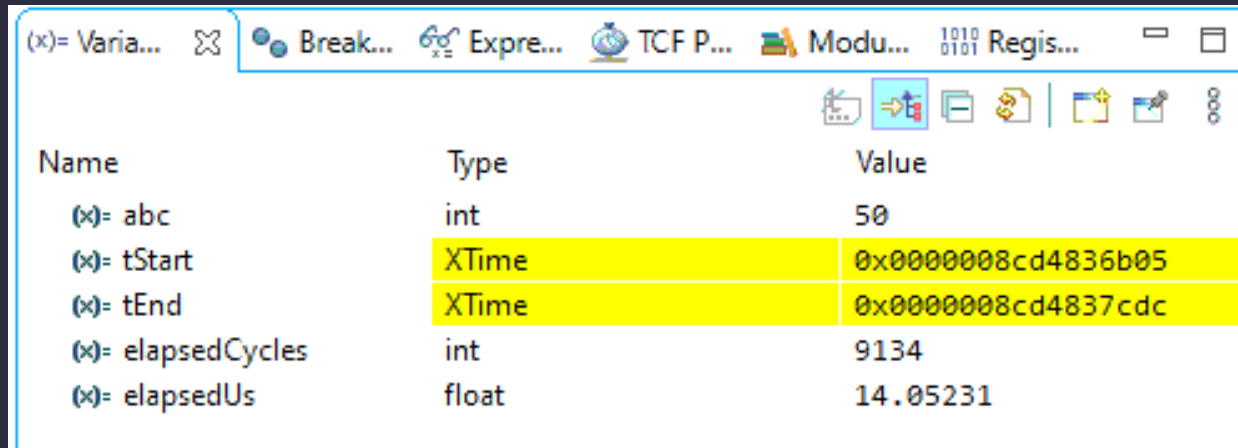
Aynı anda takip edilmek istenen birden çok değişken olduğunda, ilgili değişkene sağ tıklanıp add watchpoint'e tıklanır.

Performance Profiling and Debugging



Açılan pencerede, expression to watch bölümüne seçilen değişkenin adı yazılır.

Performance Profiling and Debugging



The screenshot shows a debugger's variables window with the following data:

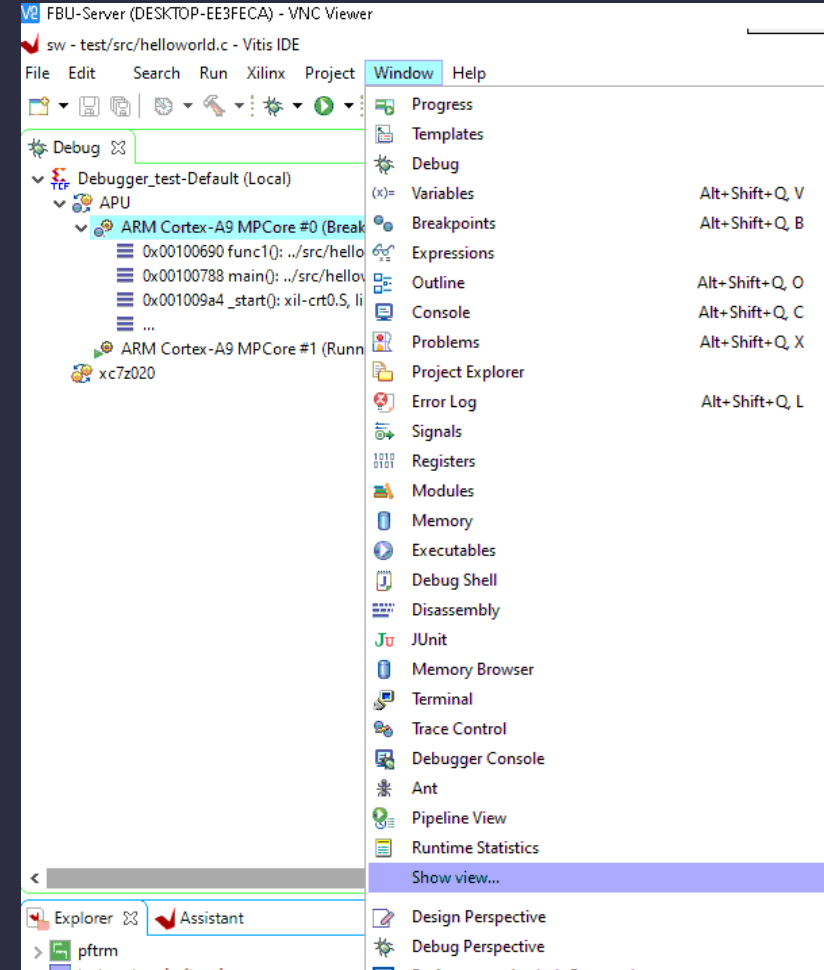
Name	Type	Value
(x)= abc	int	50
(x)= tStart	XTime	0x0000008cd4836b05
(x)= tEnd	XTime	0x0000008cd4837cdc
(x)= elapsedCycles	int	9134
(x)= elapsedUs	float	14.05231

Vitis'te sağ tarafta bulunan variables penceresinden, daha önceden eklenmiş değişkenlerin değerleri gözlemlenebilir.

Performance Profiling and Debugging

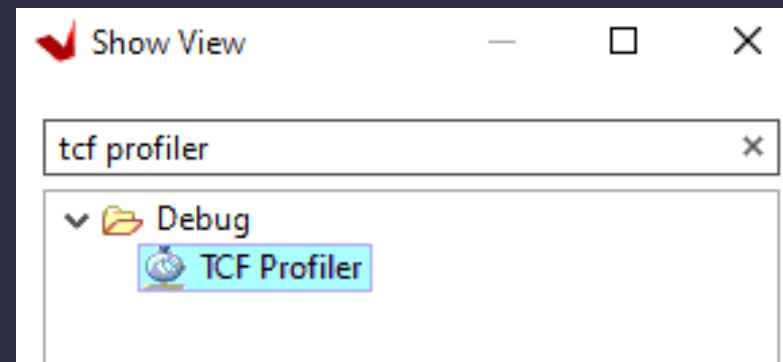
Kodun hangi satırda ne kadar zaman harcadığının anlaşılması için Profiler tool'u kullanılabilir.

Bunun için Window penceresinin altından show view'e tıklanır.



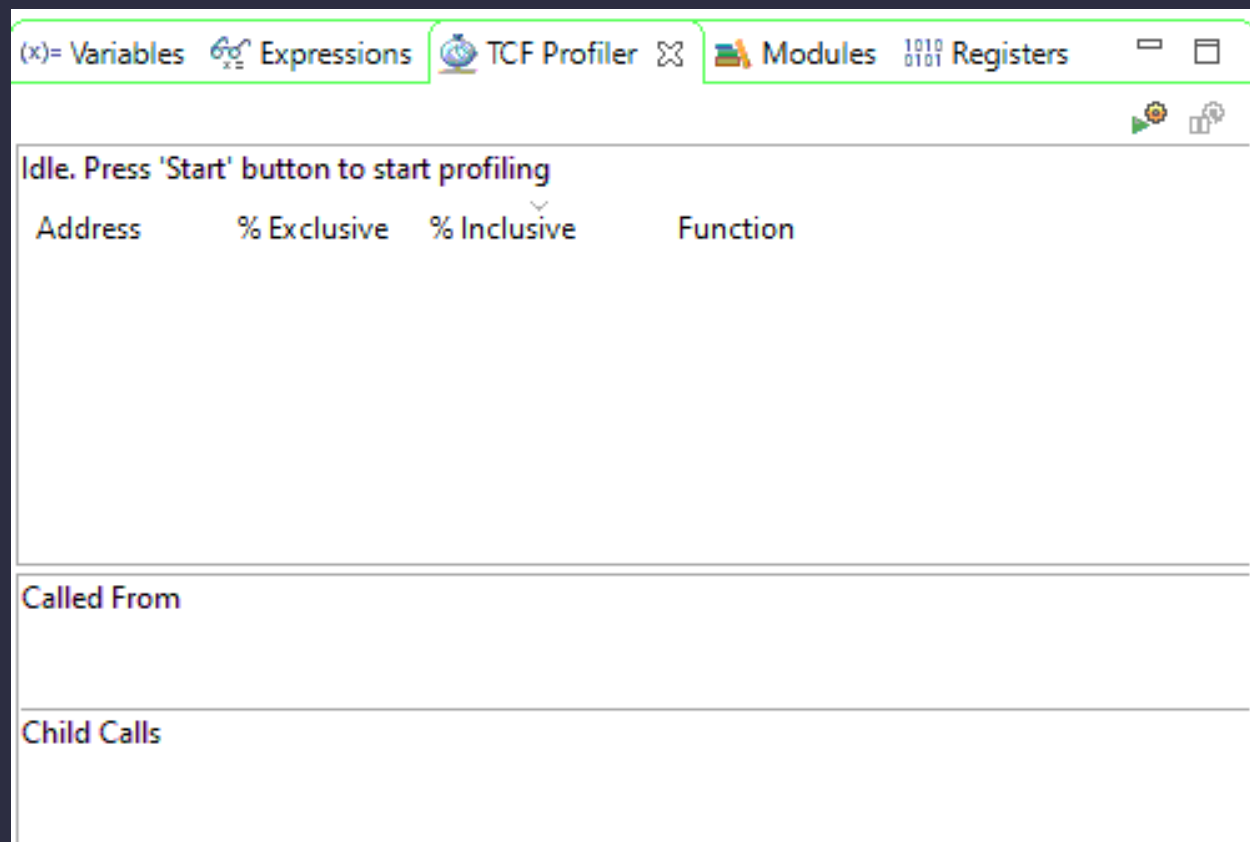
Performance Profiling and Debugging

Açılan görünümde, arama yerine TCF Profiler yazılarak, profiler tool'u açılabilir



Performance Profiling and Debugging

Açılan TCF Profiler tool'u start tuşuna basılarak çalıştırılır.

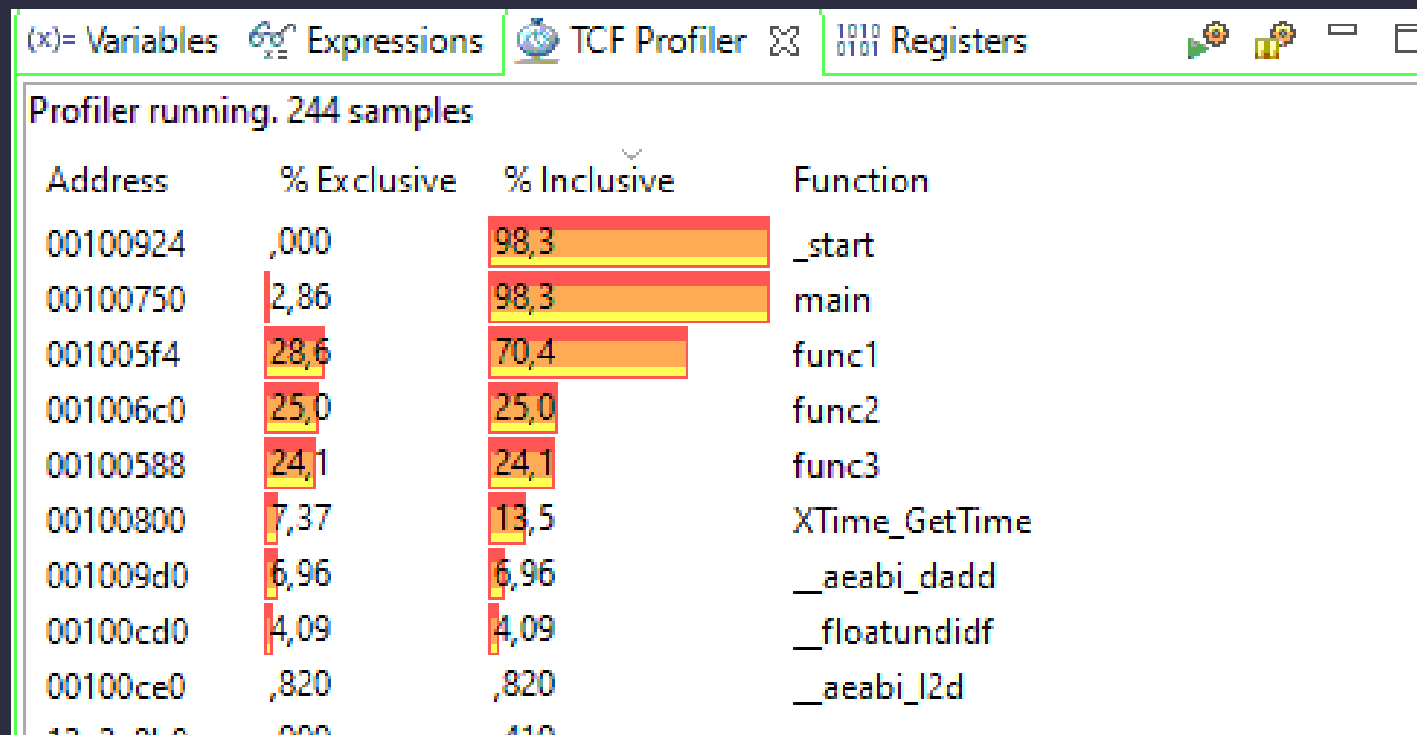


Performance Profiling and Debugging

Profiler çağrılan fonksiyonların CPU'nun yüzdeler ne kadar hesaplama gücü tükettiğini gösterir.

Exclusive sütünü, bir fonksiyonun alt fonksiyonlarının hesaplama gücü kullanımını dahil etmez.

Inclusive ise, fonksiyonun alt fonksiyonlarında hesaplama gücü kullanımının dahil edildiği gösterimdir.



Profiler running. 244 samples

Address	% Exclusive	% Inclusive	Function
00100924	,000	98,3	_start
00100750	2,86	98,3	main
001005f4	28,6	70,4	func1
001006c0	25,0	25,0	func2
00100588	24,1	24,1	func3
00100800	7,37	13,5	XTime_GetTime
001009d0	6,96	6,96	__aeabi_dadd
00100cd0	4,09	4,09	__floatundidf
00100ce0	,820	,820	__aeabi_l2d
17 2 81 0	000	410	

Performance Profiling and Debugging

İki satır arasında geçen süre hesaplanarakda profiling yapılabilir.

Bunun için

```
#include "xtime_l.h"
```

kütüphanesi kullanılabilir.

Performance Profiling and Debugging

Örnek kod parçacığı:

```
XTime tStart, tEnd;
```

```
XTime_GetTime(&tStart);
```

```
aFunction();
```

```
XTime_GetTime(&tEnd);
```

```
int elapsedCycles = 2*(tEnd - tStart);
```

```
float elapsedUs = 1.0 * (tEnd - tStart) / (COUNTS_PER_SECOND/1000000);
```

← Mikrosaniye cinsinden geçen süre