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

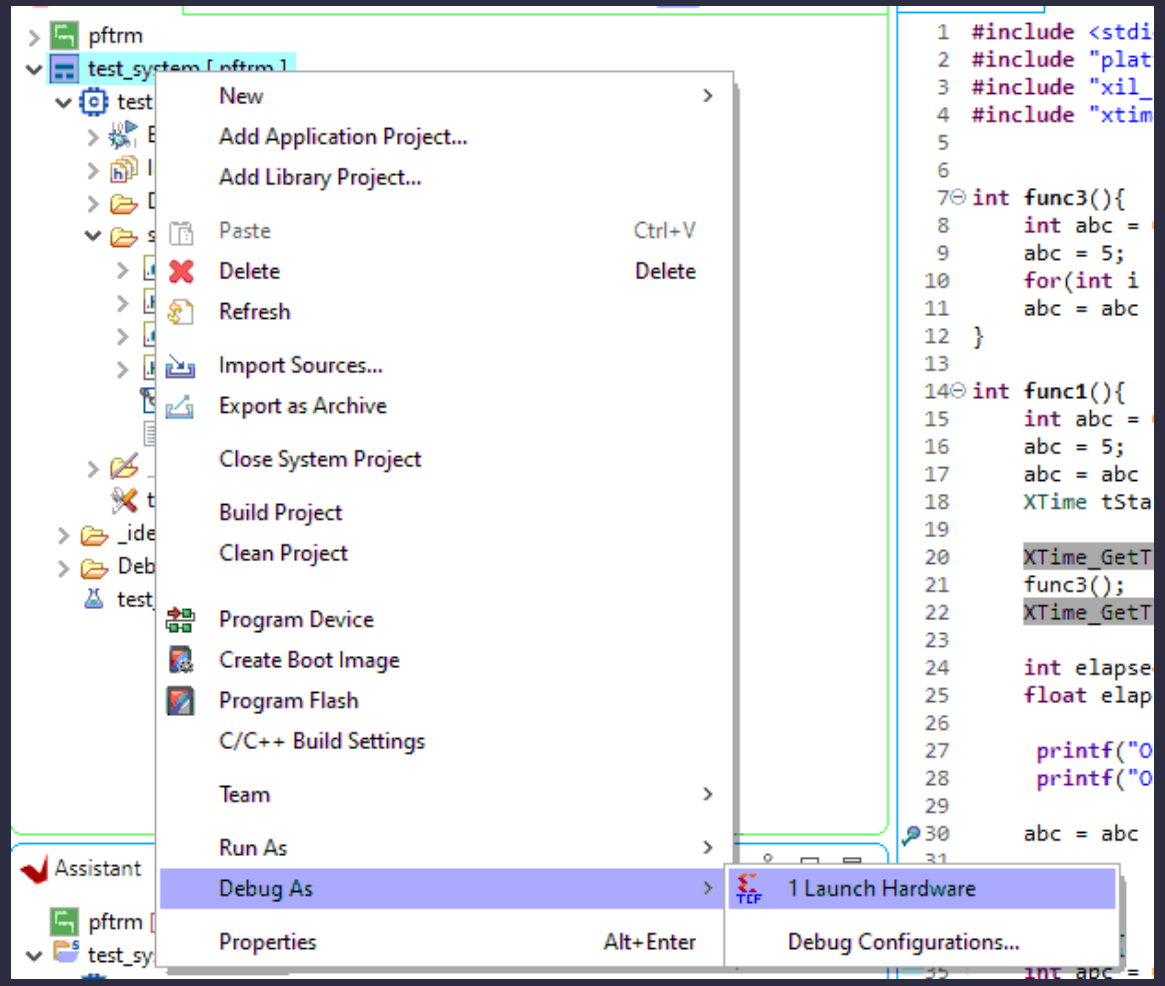
TA: Arş. Gör. Ezgi Çakmak

Office: 311

Email: ezgi.cakmak@fbu.edu.tr

Performance Profiling and Debugging

Debug başlangıcı



Performance Profiling and Debugging

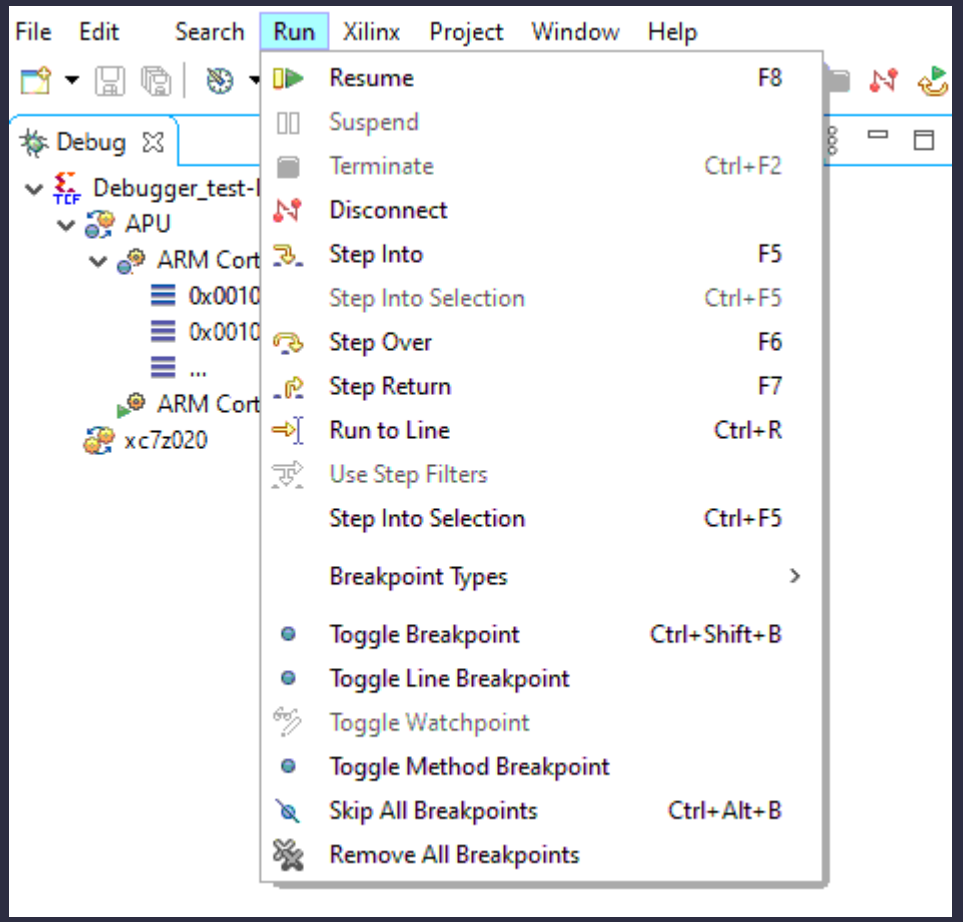
Eğer herşey doğru ise, main'in altında bulunan ilk satırdaki bulunan fonksiyonun üzerinde yeşil bir arka plan oluşacaktır.

Bu yeşil arka plan o anda beklenen satırı göstermektedir.

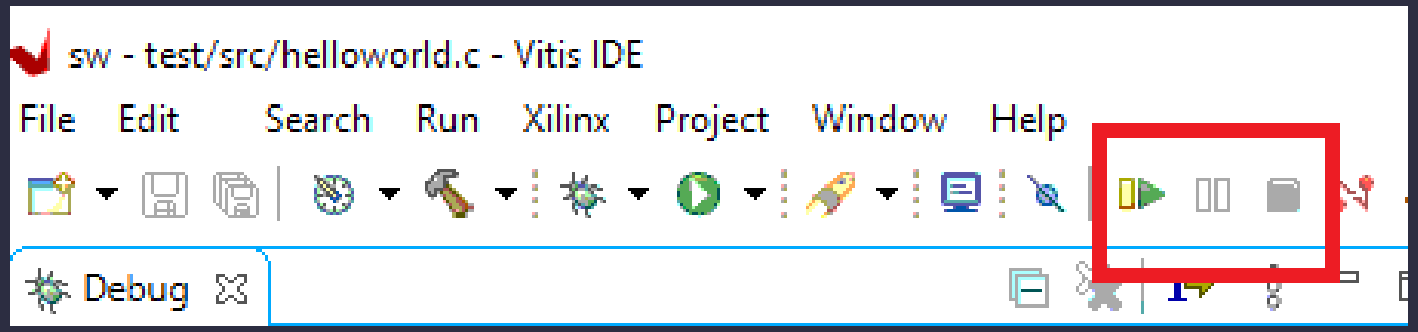
```
46  int main()
47  {
48      init_platform();
49
50
51      while(1){
52          func1();
53          func2();
54      }
55
56      print("Hello World\n\r");
57      print("Successfully ran Hello World application");
58
59
60      cleanup_platform();
61      return 0;
62  }
```

Performance Profiling and Debugging

Bu aşamada Run’un altındaki elemanlar debugging için kullanılabilir.

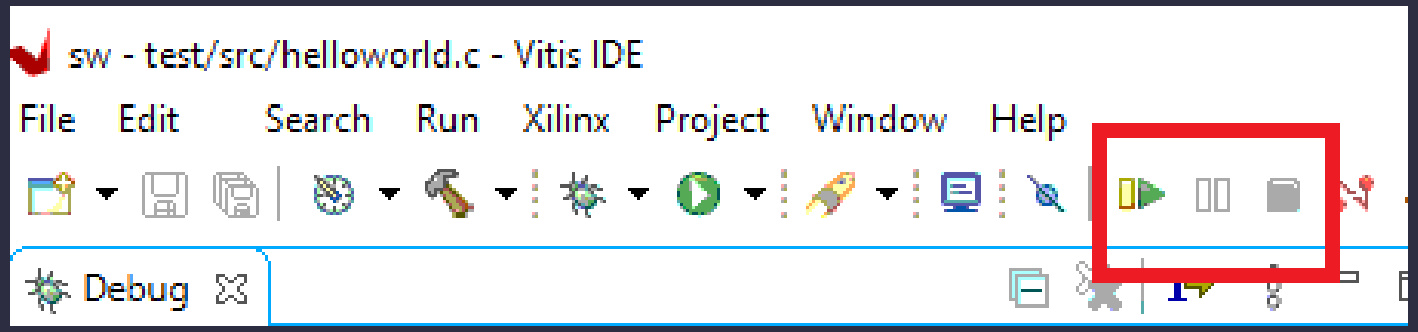


Performance Profiling and Debugging



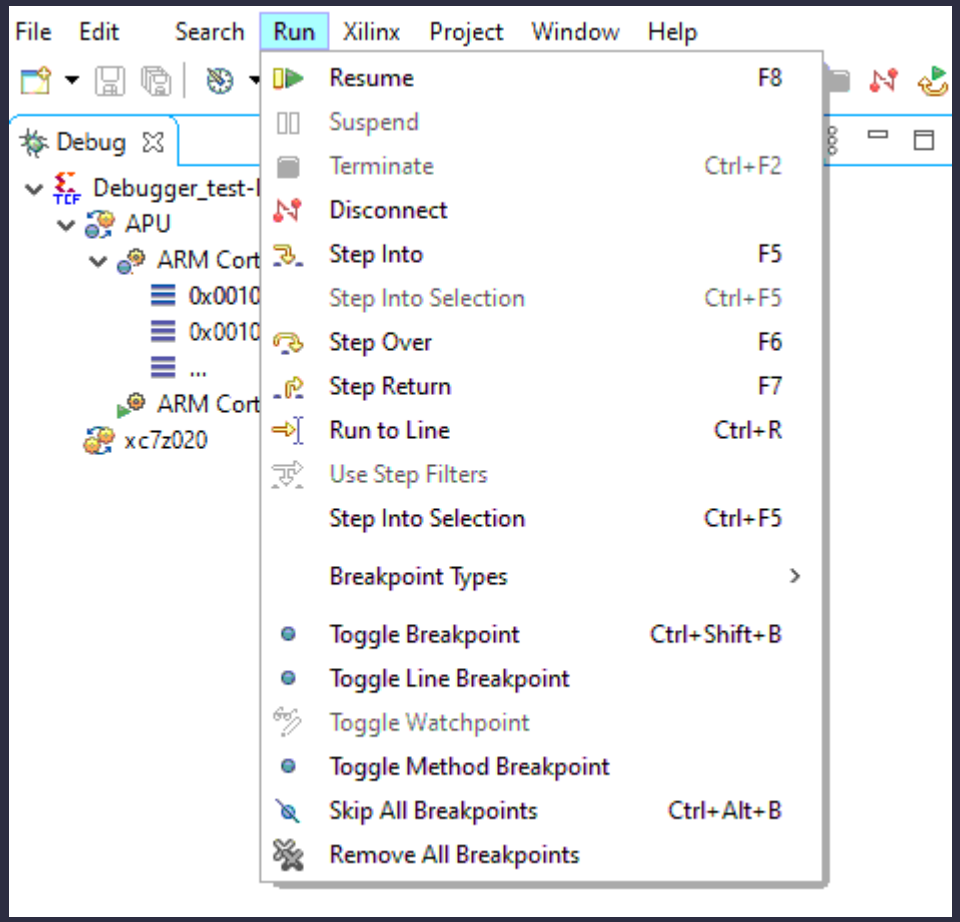
Debugging başlatıldığında kırmızı kare içerisinde Resume, Suspend ve Terminate isimli butonlar görülecektir.

Performance Profiling and Debugging



Resume, kodun yürütülmesini sağlar
Suspend, CPU o anki çalıştırdığı işi dondurur
Terminate, debugging durur

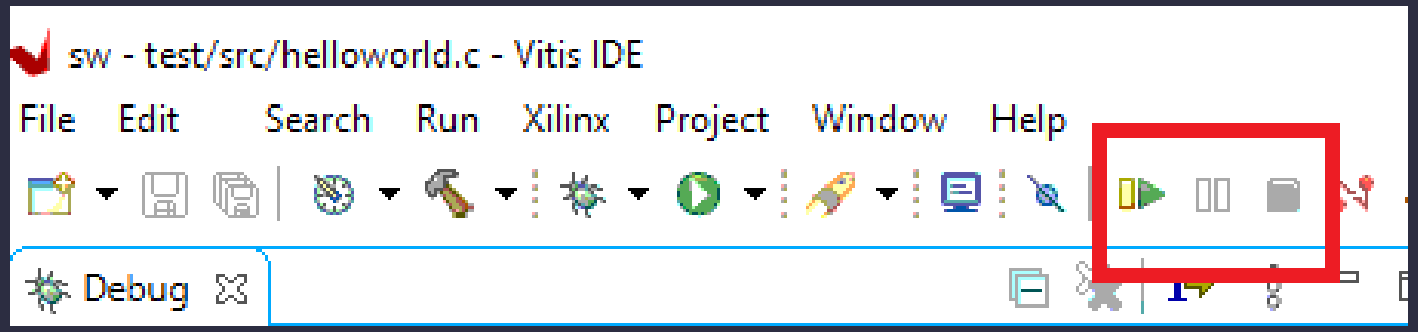
Performance Profiling and Debugging



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 ulaşılana 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

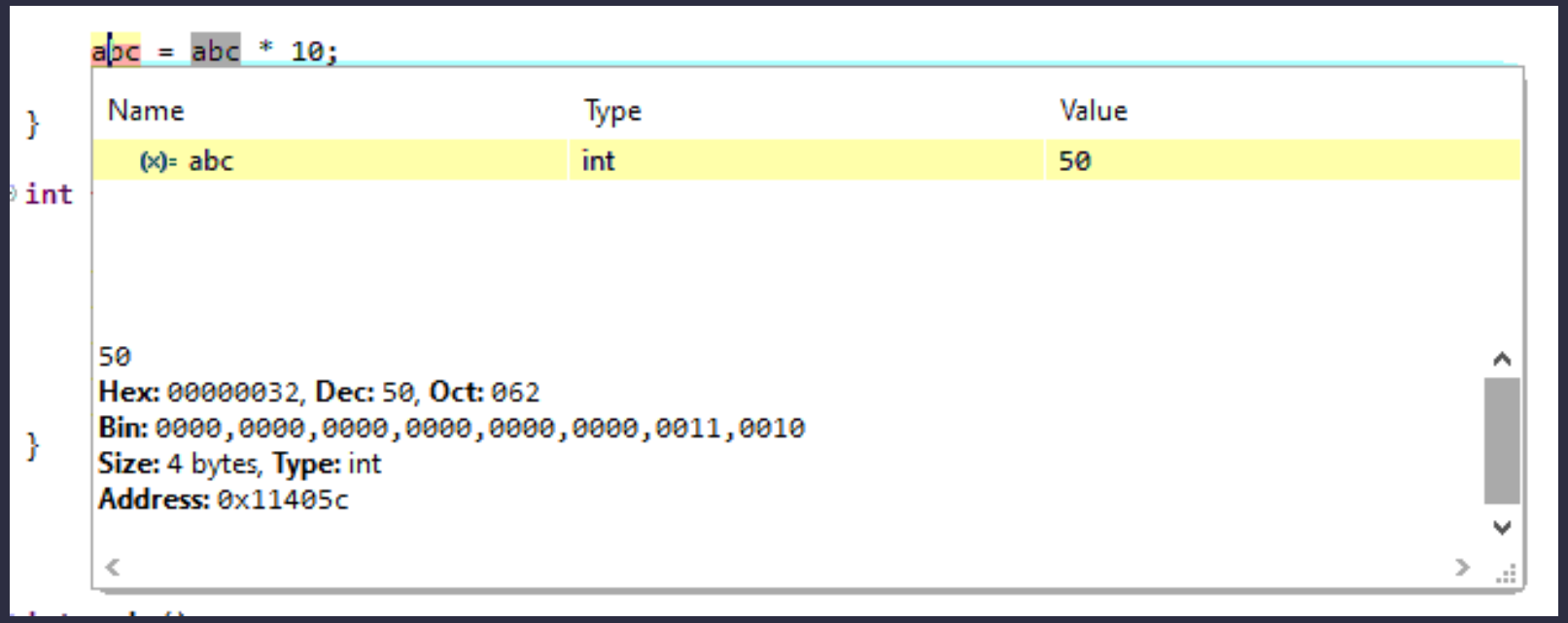
```
24      int elapsedCycles =  2*(tEnd - tStart);
25      float elapsedUs = 1.0 * (tEnd - tStart) / (COUNTS_PER_SECOND/1000000);
26
27       printf("Output took %d clock cycles.\n", elapsedCycles);
28       printf("Output took %.2f us.\n", elapsedUs);
29
30      abc = abc * 10;
31
32  }
33
34  int 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.

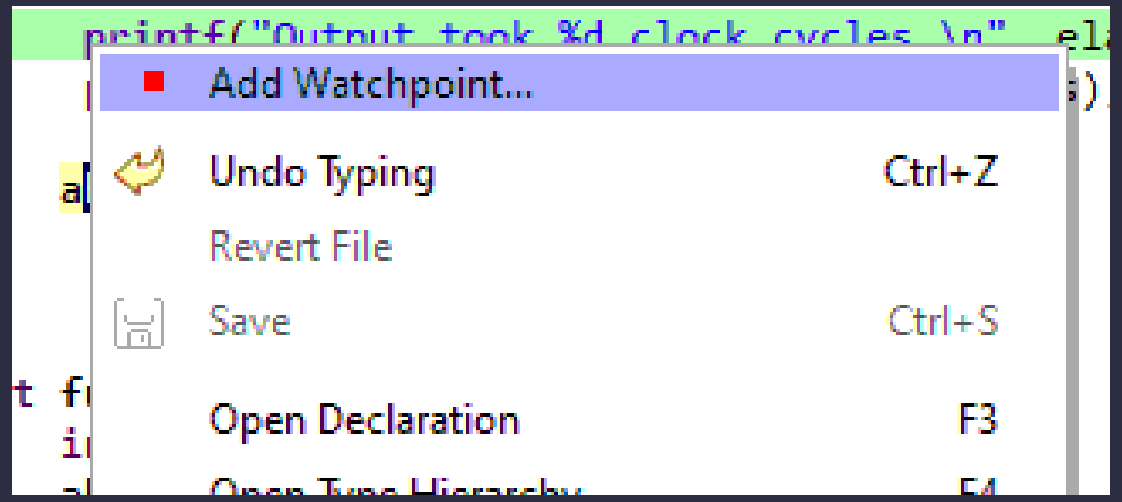
```
26
27      printf("Output took %d clock cycles.\n", elapsedCycles);
28      printf("Output took %.2f us.\n", elapsedUs);
```

Performance Profiling and Debugging



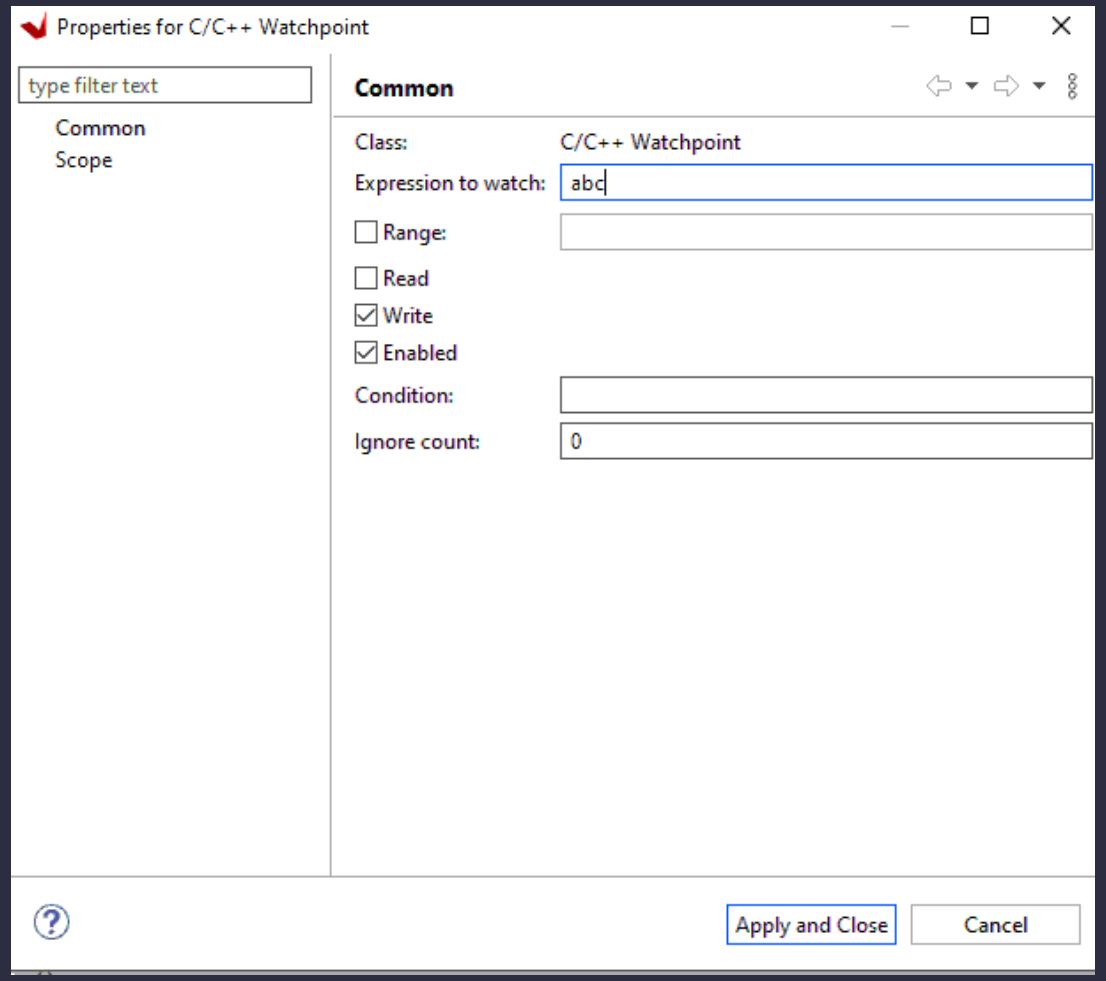
Debugging yapılırken, bir değişkenin üzerine mouse ile gelindiğinde, değişkenin o anki değeri görülebilmektedir.

Performance Profiling and Debugging



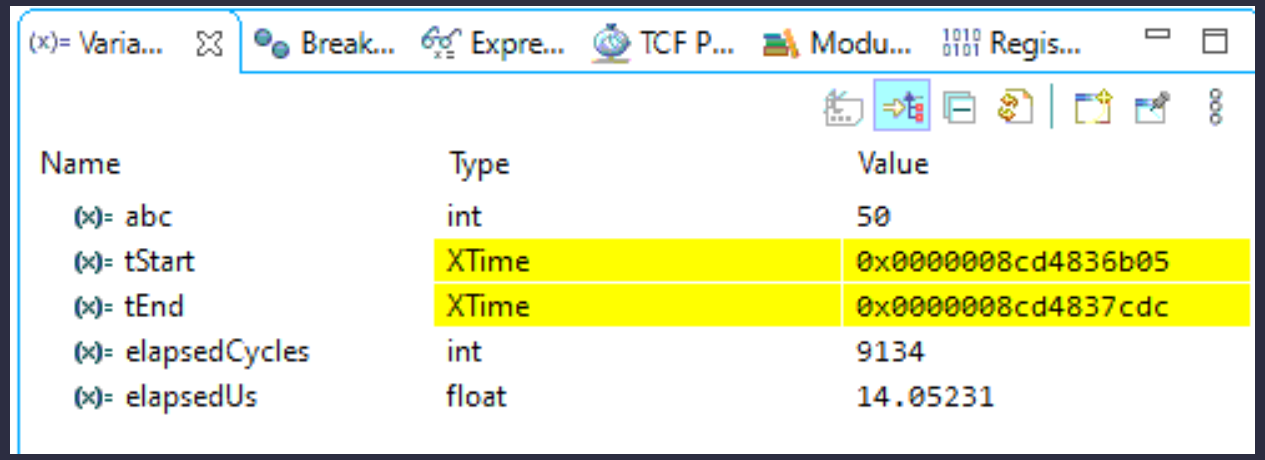
Aynı anda takip edilmek istenen birden çok değişken olduğunda, ilgili değişkene sağ tıklanıp add watchpoint'e tıklanır.

Performance Profiling and Debugging



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

Performance Profiling and Debugging

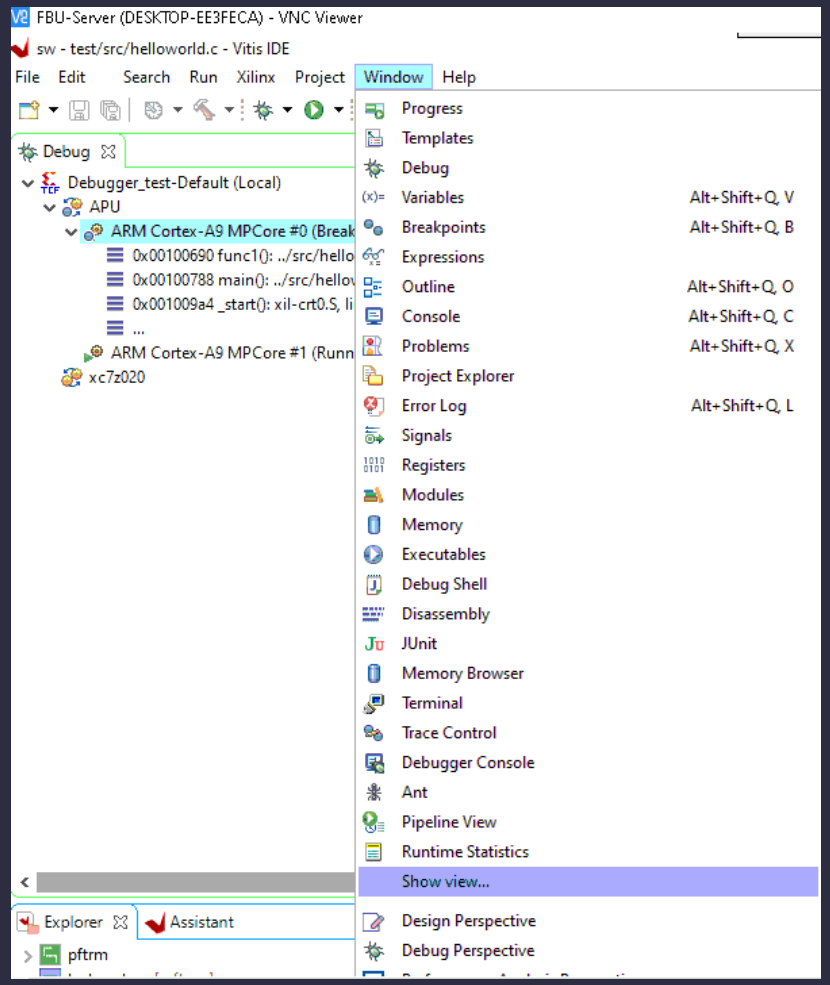


Vitis'te sağ tarafta bulunan variables penceresinden, daha önceden eklenmiş değişkenlerin değerleri gözlemlenebilir.

Performance Profiling and Debugging

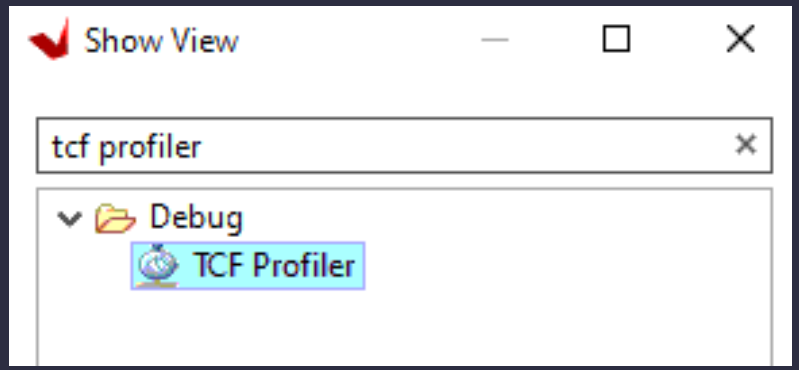
Kodun hangi satırda ne kadar zaman harcadığının anlaşılması için Profiler tool'u kullanılabilir.

Bunun için Window penceresinin altından show view'e tıklanır.



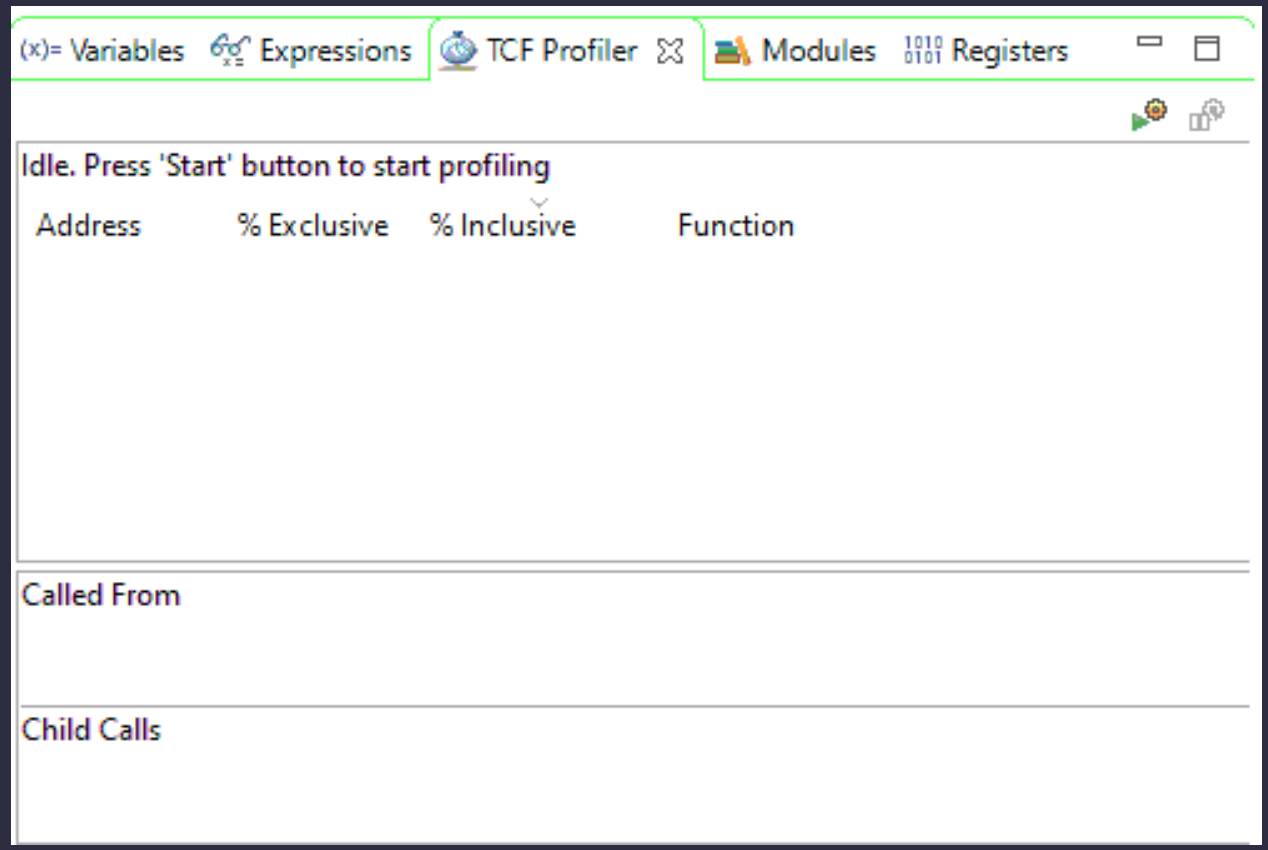
Performance Profiling and Debugging

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



Performance Profiling and Debugging

Açılan TCF Profiler tool'u start tuşuna basılarak çalıştırılır.

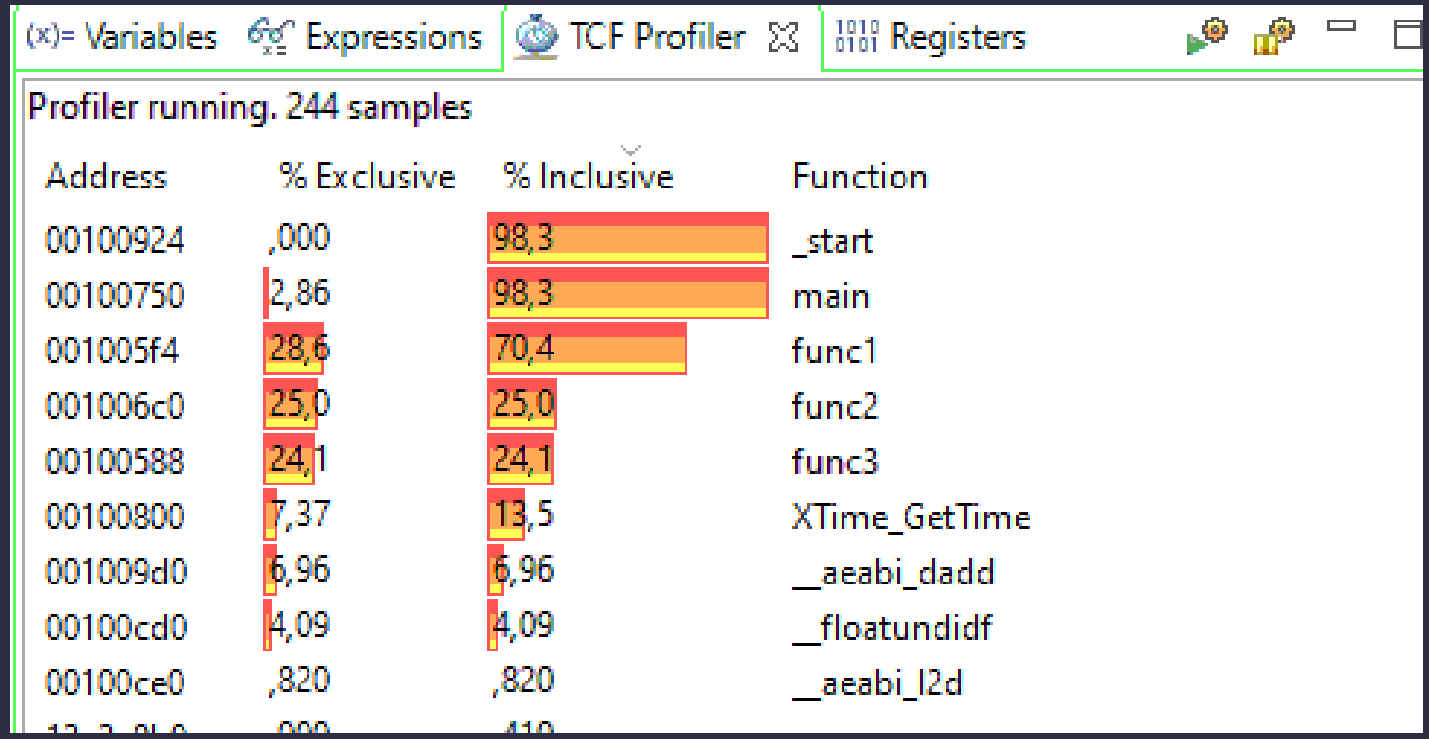


Performance Profiling and Debugging

Profiler çağrılan fonksiyonuların CPU'nun yüzdelik ne kadar hesaplama gücü tükettiğini gösterir.

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

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



Performance Profiling and Debugging

İki satır arasında geçen süre
hesaplanarakda profiling yapılabilir.

Bunun için

#include "xtime_l.h"

kütüphanesi kullanılabilir.

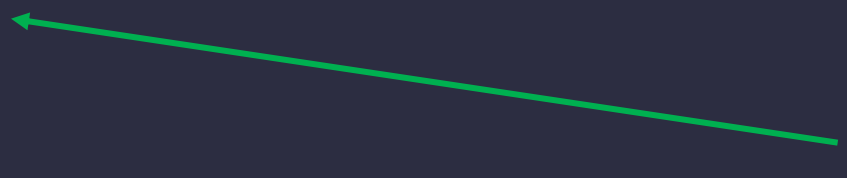
Performance Profiling and Debugging

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