IAR SYSTEMS

IAR DEVCON

EWARMの機能を効果的に使って開発効率アップ

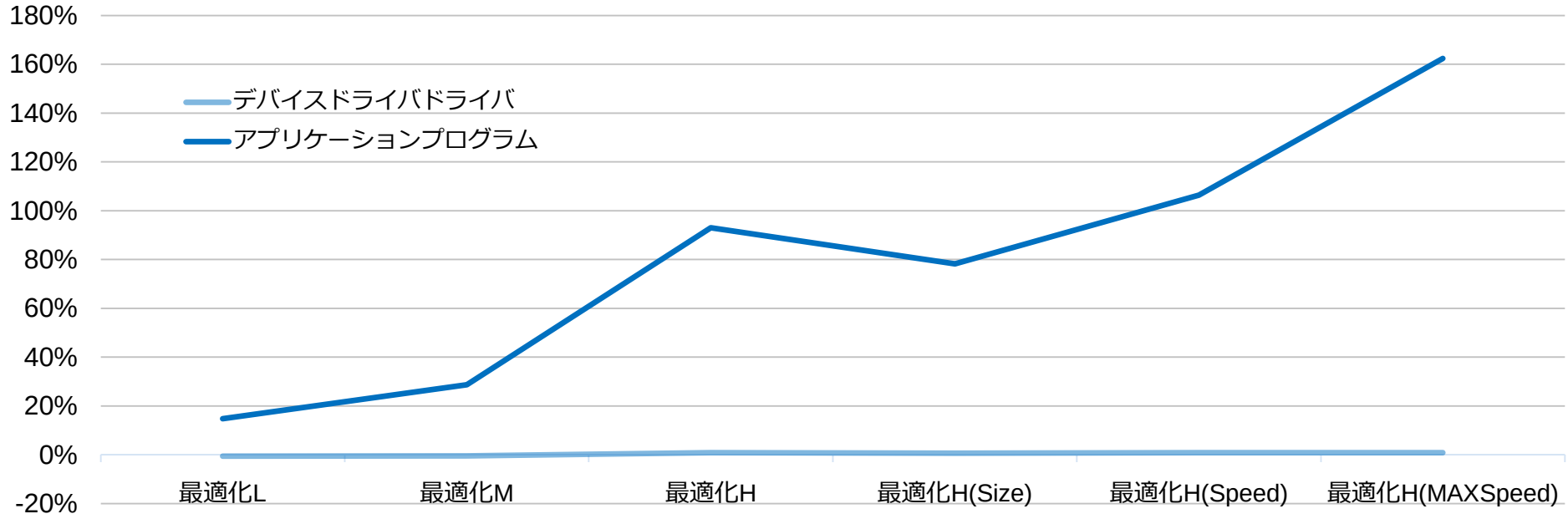
技術チーム / 殿下　信二

このセッションの目的

- IAR Embedded Workbench ® for Armを使って開発効率をアップする方法を学ぶ
  - 最適化の効果を異なるプログラムで検証
  - 最適化の基本原理を学ぶ
  - デバッグ効率とコード効率を考えた最適化の使用方法
  - 単体テストの実施方法

変化する速度最適化の効果

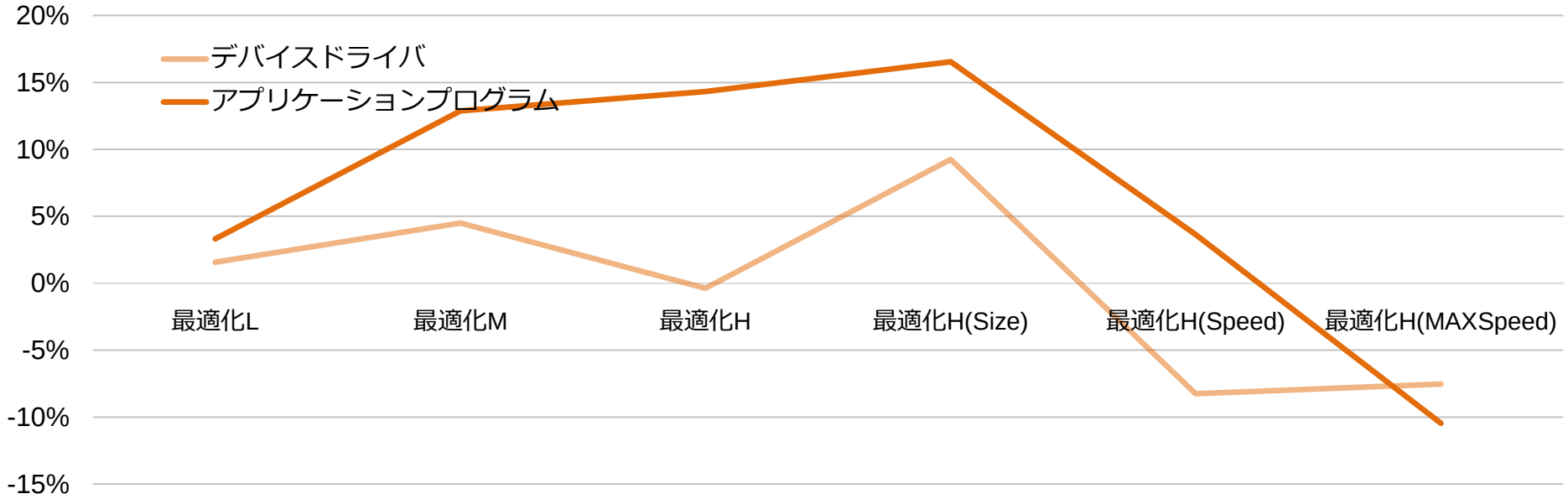
- デバイスドライバは、速度最適化の効果が小さい。



➢ 「シリアルフラッシュメモリテストプログラム」と「Coremarkベンチマーク」の最適化無しとの比較になります。

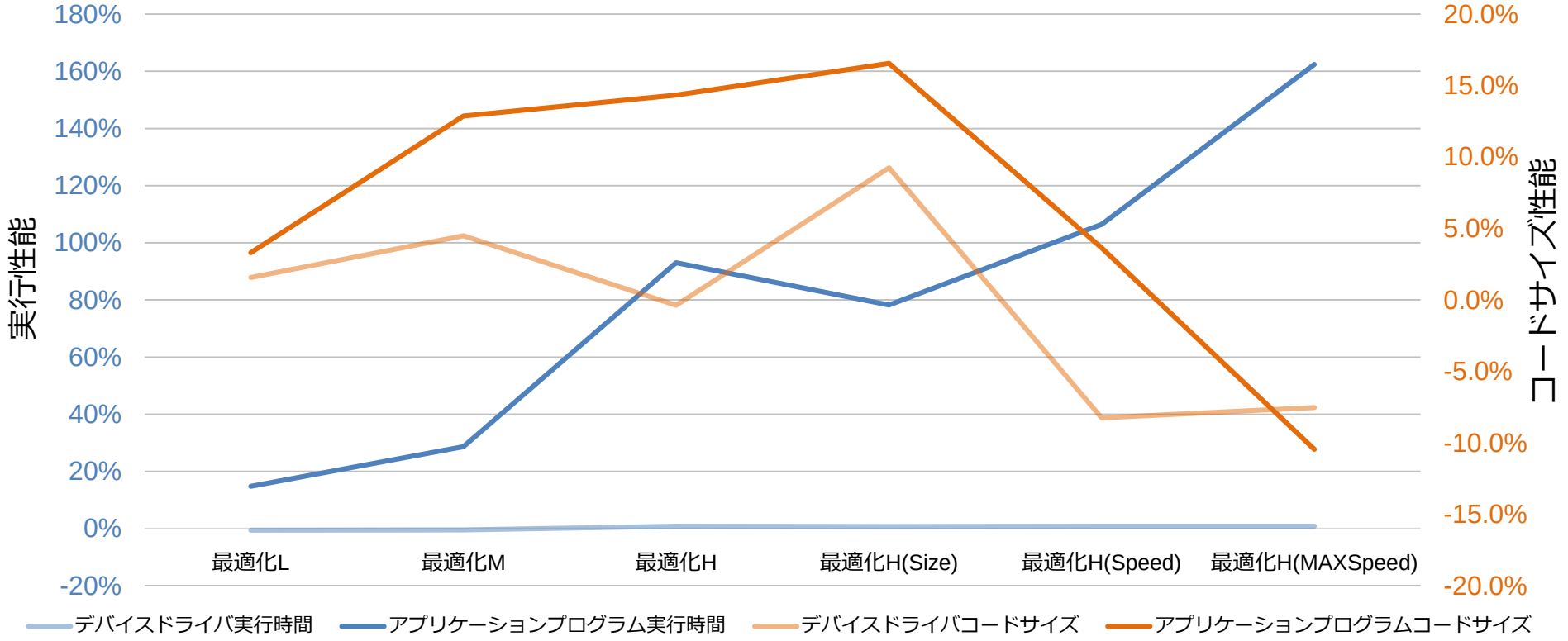
変化するコードサイズ最適化の効果

- デバイスドライバは、コードサイズ最適化の効果が小さい。



➢ 「シリアルフラッシュメモリテストプログラム」と「Coremarkベンチマーク」の最適化無しとの比較になります。

変化する最適化の効果


180%
160%
140%
120%
100%
80%
60%
40%
20%
0%
-20%
実行性能
20.0%
15.0%
10.0%
5.0%
0.0%
-5.0%
-10.0%
-15.0%
-20.0%
コードサイズ性能
最適化L
最適化M
最適化H
最適化H(Size)
最適化H(Speed)
最適化H(MAXSpeed)
デバイスドライバ実行時間
アプリケーションプログラム実行時間
デバイスドライバコードサイズ
アプリケーションプログラムコードサイズ

最適化の効果とは

- プログラムの**サイズを小さく**、**実行速度は速く**
- 演算やプログラムの流れは変化しますが、処理結果が異なる事はありません
- しかし、デバッグを行う事が難しくなる
  - 実行順序が記述順番と異なる可能性
  - 期待していた記述が無くなる可能性
  - 処理中の変数を参照する事が出来ない可能性

最適化は何をしますか？(1)

- 最適化は、共通部分をまとめ、不要なコードの削除、ループ内処理を小さく
- インライン関数による、関数呼び出しのオーバヘッドの低減や、ループ回数の最適化など

- 共通部分をまとめる

```
a=b+c*d;
e=f+c*d;
```

```
temp=c*d;
a=b+temp;
e=f+temp;
```

最適化は何をしますか？(2)

- 不要コードの削除

```
if(a>10){
  b=b*c+k;
  if(a<5)
    a+=6;
}
```

```
if(a>10){
  b=b*c+k;
}
```

- 命令をループ外に移動

```
for(i=0;i<10;i++){
  b = k * c;
  p[i] = b;
}
```

```
b = k * c;
for(i=0;i<10;i++){
  p[i] = b;
}
```

最適化を行いません(1)

- 関数呼び出しの最適化は行いません

```
int32_t func1(int32_t c,int32_t d)
{
  int32_t i;

  if(func2(c) && c){
    i = 100;
  }
  if(func2(c) || d){
    i = 200;
  }
  return( i );
}
```

```
int32_t func1(int32_t c,int32_t d)
{
  int32_t i;
  int32_t j;

  j = func2(c);
  if(j && c){
    i = 100;
  }
  if(j || d){
    i = 200;
  }
  return( i );
}
```

最適化を行いません(2)

- 構造体メンバーの順番は変えない

```
struct {
   char        a;
   int32_t     b;
   char        c;
   int32_t     d;
   int32_t     f;
   char        g;
   int32_t     h;
} data1;
```

メモリサイズ：32バイト

```
__packed struct {
   char        a;
   int32_t     b;
   char        c;
   int32_t     f;
   char        g;
   int32_t     h;
} data1;
```

メモリサイズ：20バイト
約47%性能ダウン

```
struct {
   char        a;
   char        c;
   char        e;
   int32_t     d;
   int32_t     f;
   int32_t     h;
} data1;
```

メモリサイズ：20バイト
約3.9%性能アップ

最適化によって処理が削除

- 最適化の影響は受けますか？

```
uint32_t i;

for(i=0;i<1000;i++);
```

```
```

```
extern uint32_t req;

for(;;){
  if( req==1 ){
    break;
  }
}
```

```
extern uint32_t req;

for(;;){
}
```

volatile型修飾子の機能

- 入出力ポートに相当するオブジェクト、非同期的な割込み機構によってアクセスされるオブジェクトに使用します
- 式の評価の規則で許されている場合を除いて，処理系の最適化によって削除又は順序の変更を行いません

```
volatile uint32_t i;

for(i=0;i<1000;i++);
```

```
extern volatile uint32_t req;

for(;;){
  if( req==1 ){
    break;
  }
}
```

最適化でのコード変化は？

```
ee_u16 crcu8(ee_u8 data, ee_u16 crc )
{
  ee_u8 i=0;
  ee_u8 x16=0;
  ee_u8 carry=0;

  for (i = 0; i < 8; i++){
    x16 = (ee_u8)((data & 1)^((ee_u8)crc & 1));
    data >>= 1;
    if( x16==1 ){
      crc  ^= 0x4002;
      carry = 1;
    }
    else
      carry = 0;
    crc >>= 1;
    if(carry)
      crc |= 0x8000;
    else
      crc &= 0x7fff;
  }
  return( crc );
}
```

- Coremark1.0ベンチマークソフトウェア：core_util.c:crcu8()関数

最適化を上手に使った開発は？

- 最適化レベルを上げると、デバッグ効率が悪化
- 作成コードとデバッグコードに差が発生

最適化の効果的な使用方法は

- プロジェクト全体で同じ、最適化レベルにする事は、開発効率を考えた場合、良い選択でない

- デバイスドライバは、**volatile型修飾子を使用する**ので、最適化の効果が**期待出来ない**
- **最適化設定は、無し(N)または低(L)に設定**

- アプリケーションは、**volatile型修飾子を使用しない**ので、最適化の**効果が期待出来る**
- **デバッグとリリース時で最適化設定を変更**

アプリケーションの最適化

IAR SYSTEMS

- 最適化レベルを変えても、**結果は同じ**

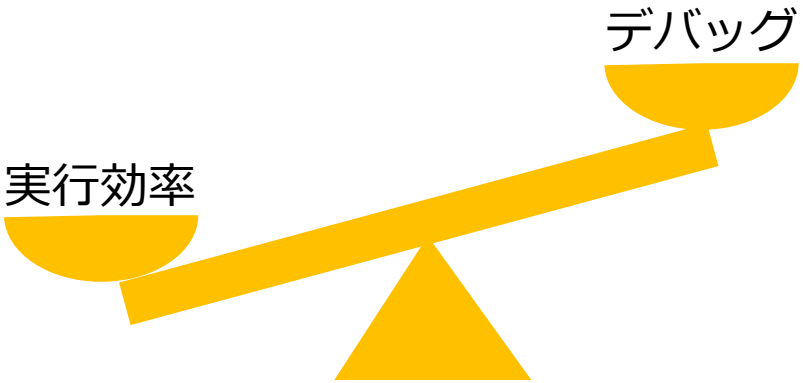
```
ee_u16 crcu8(ee_u8 data, ee_u16 crc )
{
  ee_u8 i=0;
  ee_u8 x16=0;
  ee_u8 carry=0;

  for (i = 0; i < 8; i++){
    x16 = (ee_u8)((data & 1)^((ee_u8)crc & 1));
    data >>= 1;
    if( x16==1 ){
      crc  ^= 0x4002;
      carry = 1;
    }
    else
      carry = 0;
    crc >>= 1;
    if(carry)
      crc |= 0x8000;
    else
      crc &= 0x7fff;
  }
  return( crc );
}
```

```
ee_u16 crcu8(ee_u8 data, ee_u16 crc )
{
  ee_u8 i=0;
  ee_u8 x16=0;
  ee_u8 carry=0;

  for (i = 0; i < 8; i++){
    x16 = (ee_u8)((data & 1)^((ee_u8)crc & 1));
    data >>= 1;
    if( x16==1 ){
      crc  ^= 0x4002;
      carry = 1;
    }
    else
      carry = 0;
    crc >>= 1;
    if(carry)
      crc |= 0x8000;
    else
      crc &= 0x7fff;
  }
  return( crc );
}
```

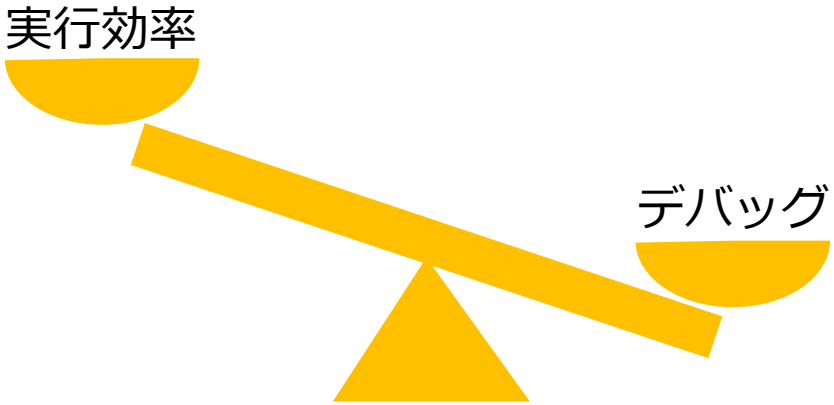
```
ee_u16 crcu8(ee_u8 data, ee_u16 crc )
{
  ee_u8 i=0;
  ee_u8 x16=0;
  ee_u8 carry=0;

  for (i = 0; i < 8; i++){
    x16 = (ee_u8)((data & 1)^((ee_u8)crc & 1));
    data >>= 1;
    if( x16==1 ){
      crc  ^= 0x4002;
      carry = 1;
    }
    else
      carry = 0;
    crc >>= 1;
    if(carry)
      crc |= 0x8000;
    else
      crc &= 0x7fff;
  }
  return( crc );
}
```

アプリケーションの最適化設定

- コーディング＆デバッグ中は、デバッグ効率を優先
- 最適化は「**無し(N)**」または「**低(L)**」を選択

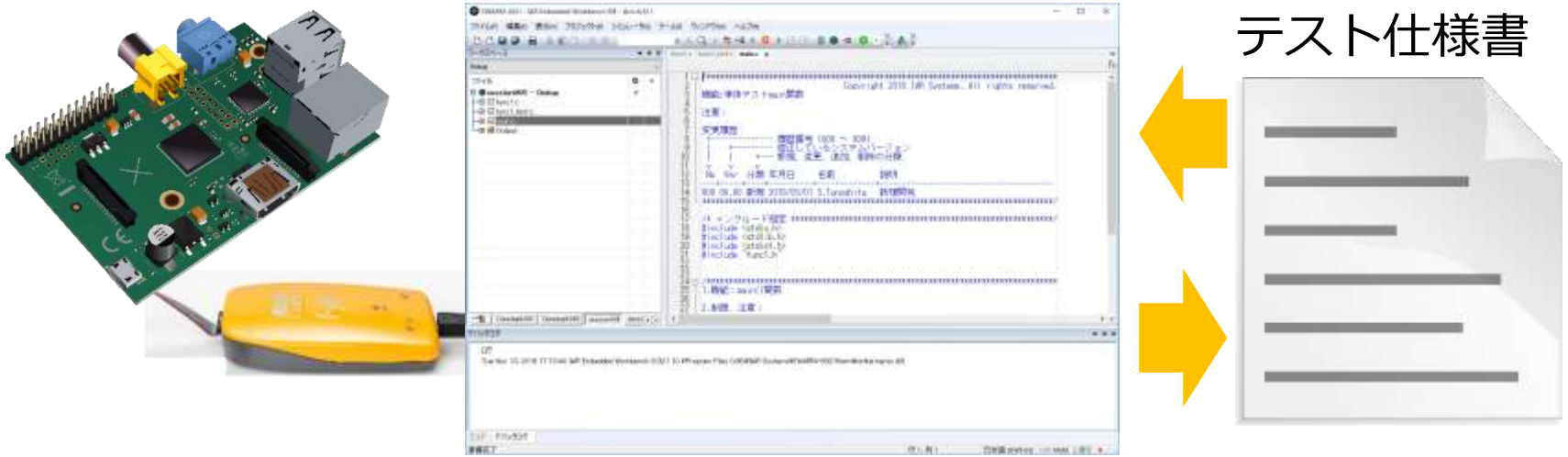


- デバッグ後は、実行性能を優先
- 最適化は「**中(M)**」または「**高(H)**」を選択



最適化設定を変えても大丈夫？

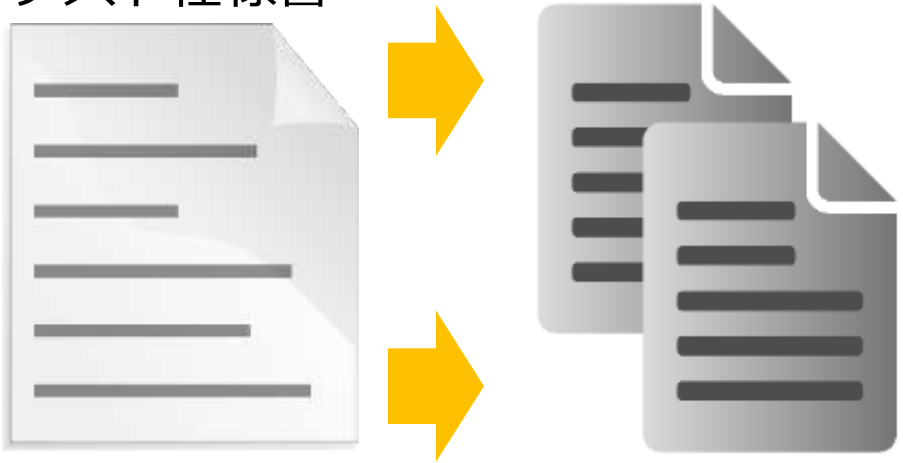
- 結果は同じですが、アセンブラ命令が異なり評価は必要
- テスト仕様書の「設定と結果」で妥当性を検証



テスト自動化を始めましょう

- 手作業での評価は、生産性を著しく低下
- ソフトウェアの資源化を阻害する大きな要因

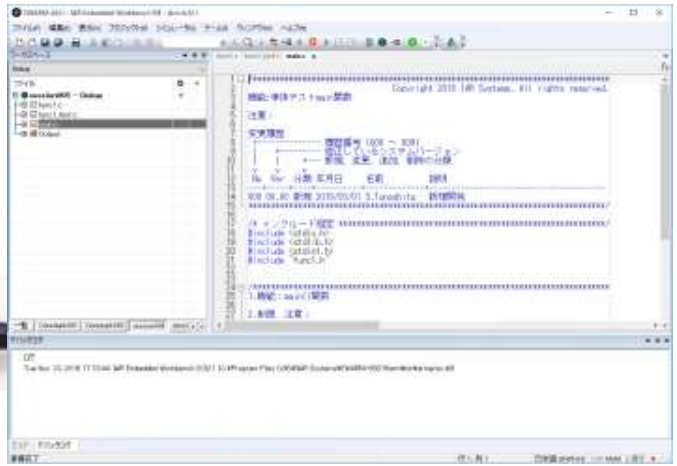
テスト仕様書



- テストパラメータ設定ファイル
- テスト結果ファイル
- ファイル形式は、CSVフォーマット

ファイル入出力を使おう

- ライブラリ選択を「フル」設定にすると、ファイル入出力を使用出来る



テストパラメータ
ファイル

テスト結果
ファイル

ソフトウェア資源化のために

- ソースコードを使用する時に安心して使える事が大切
  - テスト環境が簡単に使える事
  - テストケースが修正可能な事
  - 実機でテストが可能な事
- 4個のファイルで実現

C/C++
ソースコード

テスト
コード

テストパラメータ
ファイル

テスト結果
ファイル

まとめ

- 最適化の効果はプログラム構造で異なる
- 最適化の基本動作とvolatile型修飾子の使用方法
- IAR Embedded Workbench ® for Armファイル入出力機能で単体テストの自動化が可能

IAR DevCon Tokyo

#IARdevcon