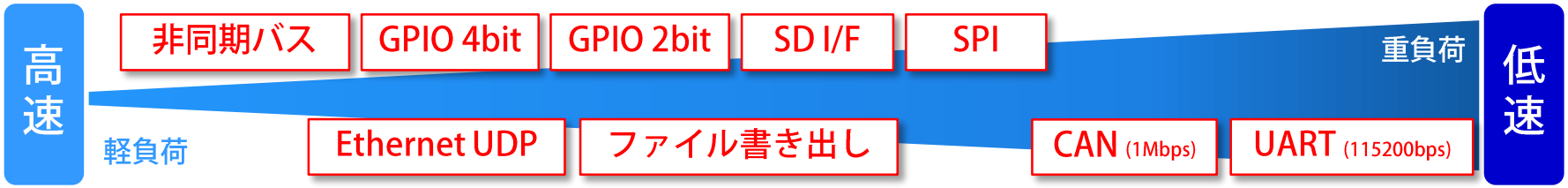
オーバーヘッドによる負荷削減ノウハウ

動的

Step 1 より高速にTPを出力できる接続方式を選択する。

- 接続方式によるオーバーヘッドの大小比較



＃動作周波数の低いCPUを使用する時は、CPUのパワーを使わないデータを転送方式を採用する。(例: SSI, DMAなど)

＃SPIの場合は、ペリフェラルを使用する。

- 高速TP機能を使用する。対象はGPIO/SPI接続。1ファイルのみ。TPは1024個限定。

Step 2 使用目的に合わせて、TP挿入箇所を工夫する。

- 通過情報だけでも確認したい場合は、FuncIn/FuncOutのみTPを挿入する。
- 複数のテスト結果を合算して解析できるので、テスト毎にTP挿入箇所を制限する。

Step 3 ドライバのカスタマイズによるオーバーヘッド削減。

- アセンブラを使用して、ドライバを高速化する。
- 用途に応じた特殊カスタムドライバの活用。

＃カバレッジ向け自己ワンタイムトレース。　＃割込み処理内のトレースを可能とする外部トリガーによるデータ出力制御。

＃FIFOに溜めたデータを低優先度のデータ出力タスクで出力制御するFIFOバッファ制御。 etc.

HEARTLAND.DATA

カスタム例: 自己ワンタイムトレース

動的

概要

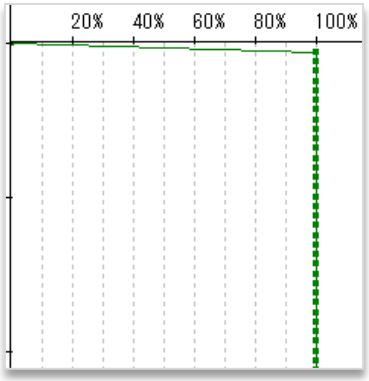
DT10の機能であるワンタイムトレースターゲット側自身で実施

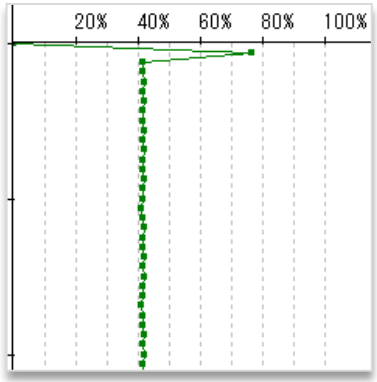
方法

- ターゲット側で通過情報を保持
- 新規に通過したTPだけデータ出力

メリット/デメリット

- 高負荷な状態でもC0カバレッジの取得が可能
- パフォーマンス測定には向かない





ドライバのコード例

```
#define MAX_TEST_POINT200
#define MAX_COVERAGE_BUF_SIZE ((MAX_TEST_POINT+3)/4)
static unsigned int coverageBit[MAX_COVERAGE_BUF_SIZE];

void _TP_BusOut( DT_UINT addr, DT_UINT dat )
{
   unsigned int key;
   portInit();
   enterCritical();
   if( addr == 0 ){
      _TP_BusOutDrv( addr, dat );
   }
   else{
      key = ((addr >> 4) << 5) + (dat & 0x1f);
      if( coverageBit[key >> 5] & (1 << (key & 0x1f))){
         /* Already Passed */
      }
      else{
         _TP_BusOutDrv( addr, dat );
         coverageBit[key >> 5] |= (1 << (key & 0x1f));
      }
   }
   exitCritical();
}
```

カスタム例: 外部トリガーによるデータ出力

動的

概要

常にデータを出力するのではなく、外部トリガーに応じてデータ出力をON/OFFする

方法

- ある端子がHIのときにTPの通過情報を内部メモリに溜める
- 端子がLOWになったトリガーで、溜めたデータを一気に吐き出す

メリット/デメリット

- 割り込み処理内のトレースが可能
- パフォーマンス測定には向かない
- 通過情報を溜めるための一定量のメモリが必要

ドライバのコード例

```
#define MAX_BUF_SIZE 2048
static unsigned int testPointBuff[MAX_BUF_SIZE];
static unsigned int testPointCount;

void _TP_BusOut( DT_UINT addr, DT_UINT dat )
{
   unsigned int key;
   unsinned int i;
   static int sw = 0;

   portInit();
   enterCritical();
   if( sw == HI && input_sw() == LOW ){
      for( i = 0; i < MAX_BUF_SIZE; i++ ){
         key = testPointBuff[i] ;
         addr = key >> 16;
         dat = key & 0xffff;
         _TP_BusOutDrv( addr, dat );
      }
      testPointCount = 0;
   }
   else if( input_sw() == HI ){
      if( testPointCount  < MAX_BUF_SIZE ){
         key = addr << 16 + dat;
         testPointBuff[testPointCount ++] = key;
      }
   }
   sw = input_sw();
   exitCritical();
}
```

カスタム例: オーバーヘッドを極限に抑える

動的

概要

FIFOバッファを作成し、テストポイント通過時はFIFOへデータを書き込むだけとする

方法

- FIFOバッファを作成
- 溜まったデータはデータ出力タスクにて出力
- データ出力タスクの優先度は低めに設定

メリット/デメリット

- 割り込み処理のトレースが可能
- 一定期間低負荷状態が続くシステムで長時間のトレースが可能
- FIFOがフルになったときは、FIFOに空きができるまで待つかデータを捨てる必要がある

FIFOイメージ