UART1 のコード生成されたコードのシミュレータでの動作確認(改)

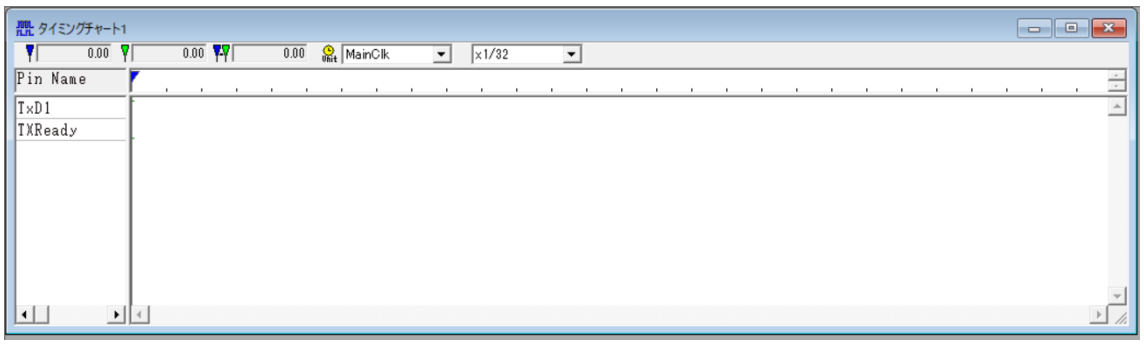
コード生成された通信用 API の問題点の一つが、通信完了の処理は callback 関数でユーザーが処理する必要があることです。普通は通信状態を示すフラグを準備しておくのですが、生成されたコードにはこのハンドリング処理が全くありません。普通であれば、通信を開始したら、フラグをクリアし、通信が完了(指定されたデータが全て送信完了)したら、フラグをセットすることになります。しかし、callback だけではフラグのセットしかできません。そこで、フラグのクリアは main 処理のレベルで実行することになります。

普通は、フラグは変数として、メモリ上に確保するのですが、今回のスレッドでは「8 バイト送信したいのに 1,2 バイトしか送信できない」となっていたので、メモリではなく、SFR(具体的に出力ポート)を使って、外部から確認できるようにしてみました。出力ポートの初期状態は 0 なので、それをそのまま使うことにします。ここでは、P30 を使います。20pin の RL78/G13 は持っていないので、ここではシミュレータのタイミングチャート機能を使って確認することにします。

測定に使用する main プログラムを以下に示します。

```
70                  void main(void)
71                  {
72      0019e           R_MAIN_UserInit();
73                      /* Start user code. Do not edit comment generated here */
74                      {
75
76                  //      uint8_t work;
77
78      001a1           P3_bit.no0 = 0x00;                  /* 送信完了フラグをクリア   */
79
80                      while( 1 )
81                      {
82
83      001b0               R_UART1_Send( (uint8_t *)g_tx_buff, 0x08 );     /* 送信処理開始  */
84      001a7               while( 0 == P3_bit.no0 )            /* 送信完了待ち             */
85                          {
86      001a6                   NOP();
87                          }
88
89
90      001ac               NOP();
91
92      001ad               P3_bit.no0 = 0x00;                  /* 送信完了フラグをクリア   */
93
94                      }
95                  }
96                  /* End user code. Do not edit comment generated here */
97              }
```

90 行目の NOP();はブレークポイントを設定するために準備したものです。下に GUI のタイミングチャート部分を示します。上が TxD1 信号で、下側の信号が P30 です。



r_cg_serial_user.c については、r_uart1_callback_sendend 関数の所に次に示すように、

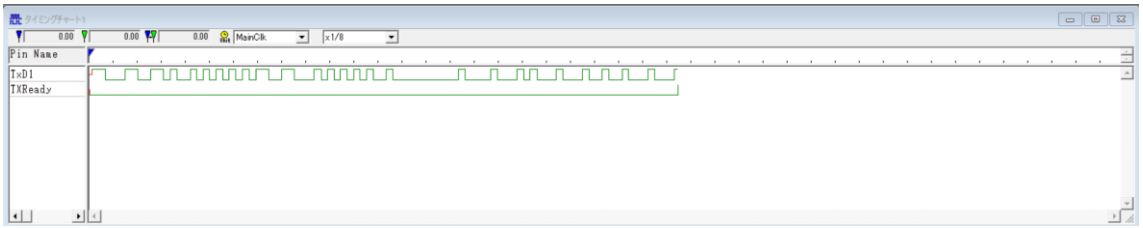
P3_bit.no0 = 1;を追加するだけです。

```
84 static void r_uart1_callback_sendend(void)↓
85 {↓
86     /* Start user code. Do not edit comment generated here */↓
87      P3_bit.no0 = 1;↓
88     /* End user code. Do not edit comment generated here */↓
89 }↓
```

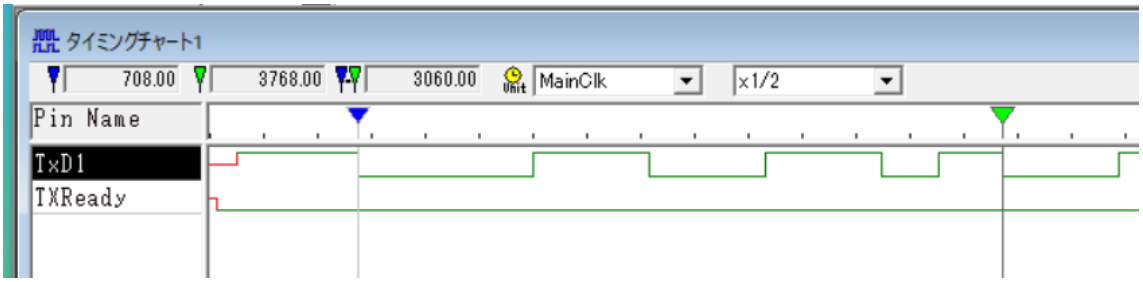
また、R_MAIN_UserInit 関数は R_UART1_Start()関数を呼び出して UART1 を起動しています。

```
105 void R_MAIN_UserInit(void)↓
106 {↓
107     /* Start user code. Do not edit comment generated here */↓
108     R_UART1_Start();              /* UART1動作起動              */↓
109     EI();↓
110     /* End user code. Do not edit comment generated here */↓
111 }↓
```

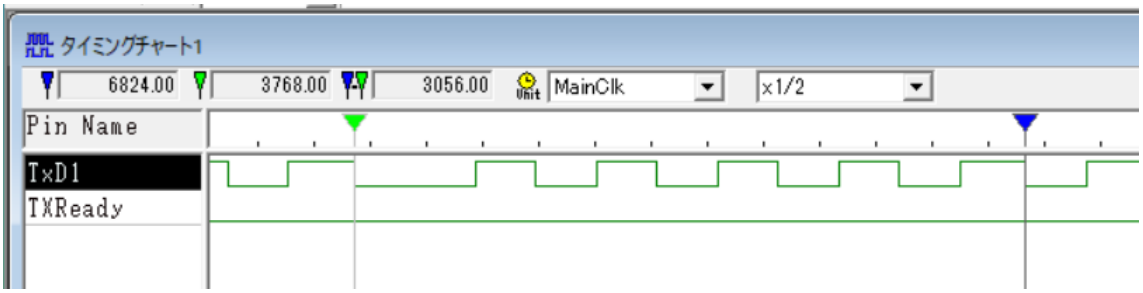
シミュレータにダウンロードして、ブレークポイントありで実行させると以下のようになります。



最初のスタートビットからストップビットまでは以下のようになります。データは 0x33 なので、偶数パリティなので、パリティビットは 0 になりその後ストップビットで終わります。

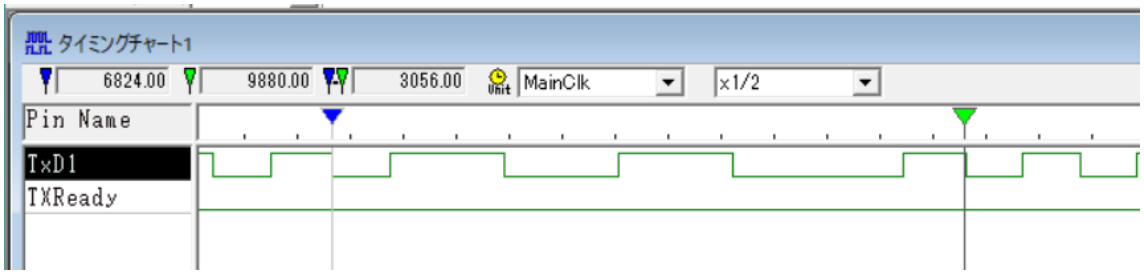


2 バイト目は以下のようになります。データは 0x55 なので、このような波形になります。

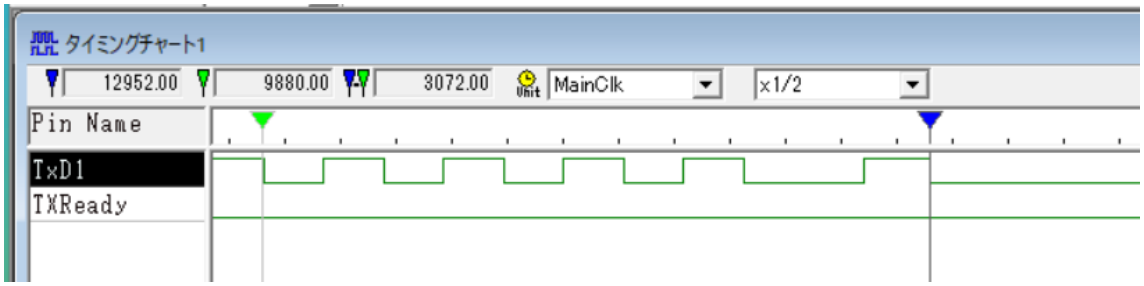


3 バイト目は次のようになります。データは 0xCC なので、ここでもパリティは 0 になっていま

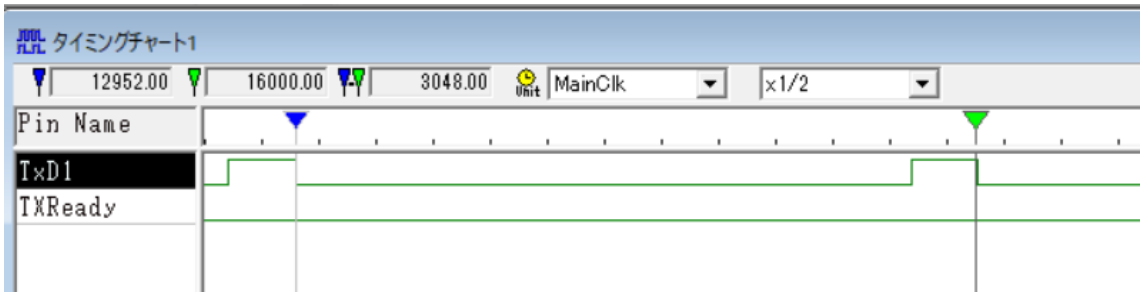
す。



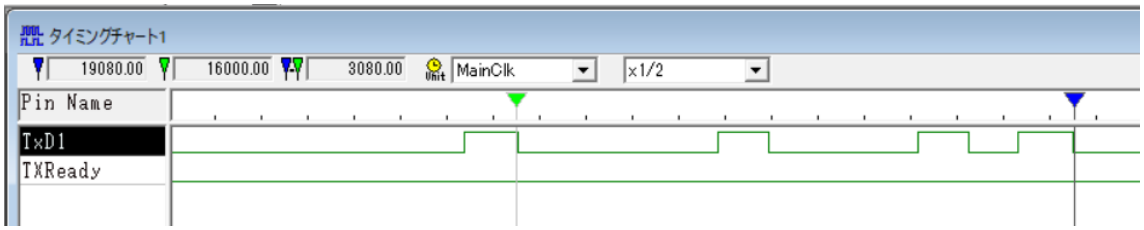
次のデータは 0xAA なので、次のような波形になっています。



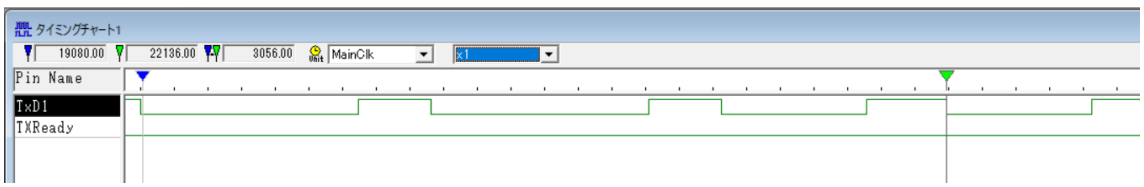
次のデータは 0x00 なので、スタートビットからパリティビットまで 10 ビット分 0 となります。



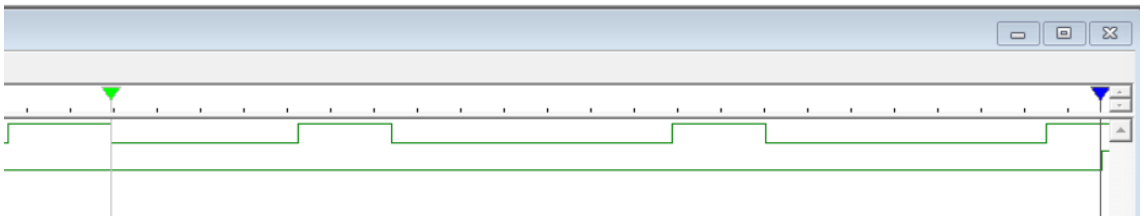
次のデータは 0x11 なので、4 ビット 0、1 ビット 1、3 ビット 0, パリティが 0、最後がストップビットになっています。



次(7 バイト目)は、0x22 なので、3 ビット 0、1 ビット 1, 3 ビット 0, 1 ビット 1、2 ビット 0 となっています。



最後は 0x44 になります。0 の状態は 2 ビット、3 ビット、3 ビットとなっています。

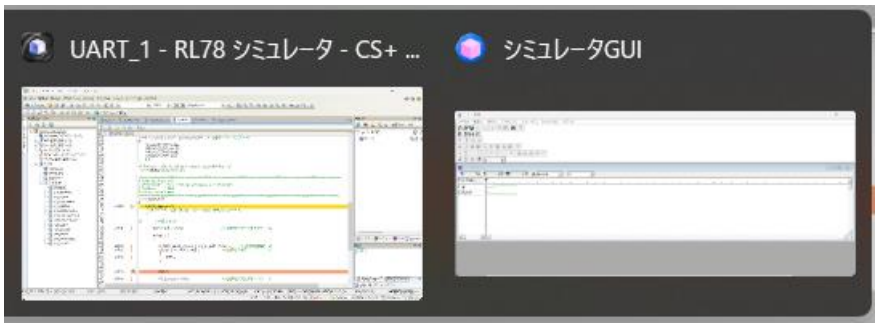


このように、1,2 バイトでなく 8 バイト分が送信されていることが確認できます。また、青いマーカーのところで、P30 が立ち上がり、ブレークがかかることで、ここまでの信号変化をタイミングチャートとしてキャプチャしています。

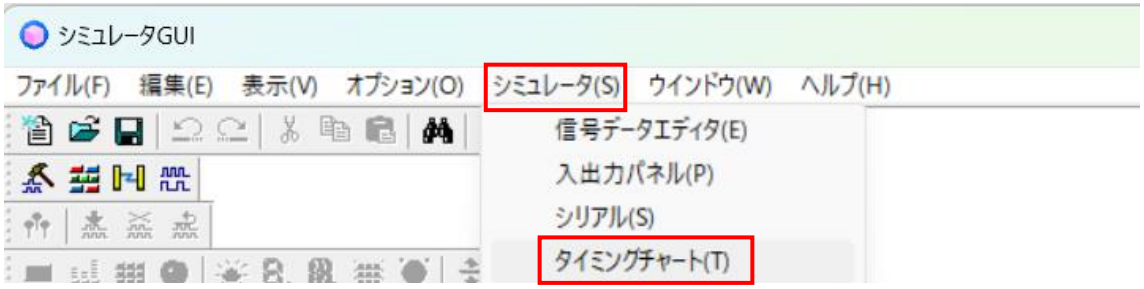
通常のやり方では通信完了のフラグはメモリ上の変数を使います。今回は、送信データとの関係を波形で見られるようにするために、あえて P30 を使用しています。RL78 としては、メモリ空間の変数も、SFR 空間のポートもアドレスが異なるだけです。違いは、メモリのハード的な初期値は不定ですが、CC-RL で初期化されます。それに対して、P30（出力ラッチ）はハード的にクリアされています。

おまけ

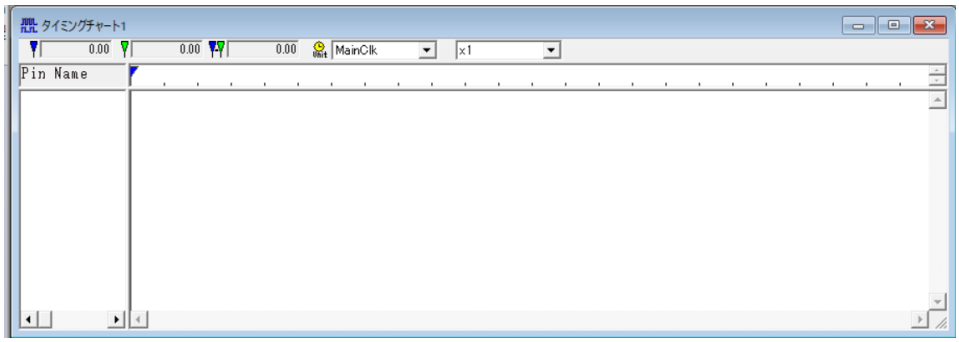
プロジェクト・ツリーでデバッグ・ツールをシミュレータに変更して、ビルドした結果をダウンロードすると下に示す 2 つのウィンドウが表示されます。



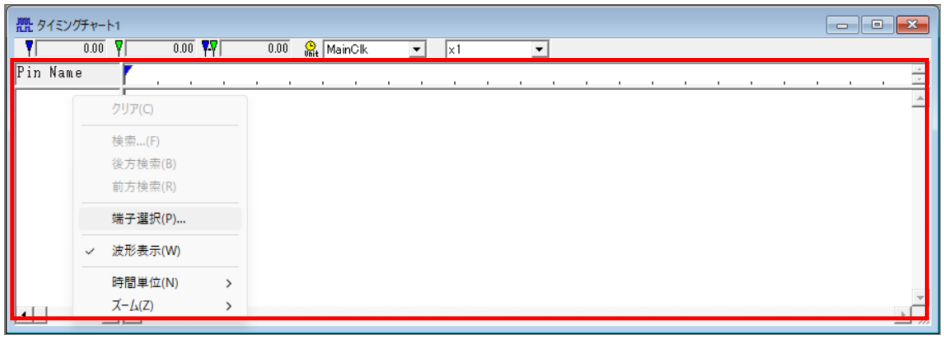
今回は、シミュレータのタイミングチャート機能を使用するので、シミュレータ GUI ウィンドウを選択します。「シミュレータ(S)」のプルダウン・メニューから「タイミングチャート(T)」を選択します。



すると、以下のサブウィンドウが作成されます。



マウス・カーソルを赤で囲んだ領域に合わせて、右クリックするとサブメニューが表示されるので、「端子選択(P)」を選択します。

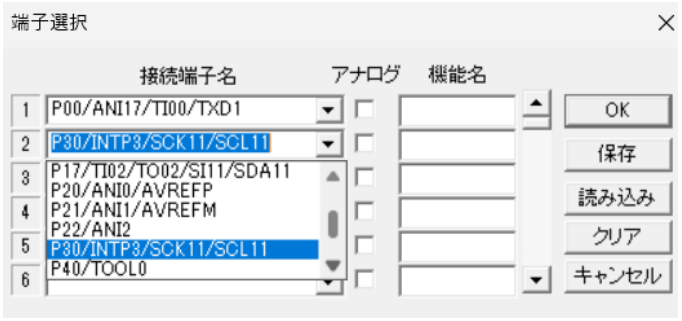


すると、以下に示す設定画面が表示されます。

そこで、「接続端子名」の「1」の右の ▼ をクリックすると、接続端子のリストが表示されるので、

「P00/‥‥/TXD1」を選びます。

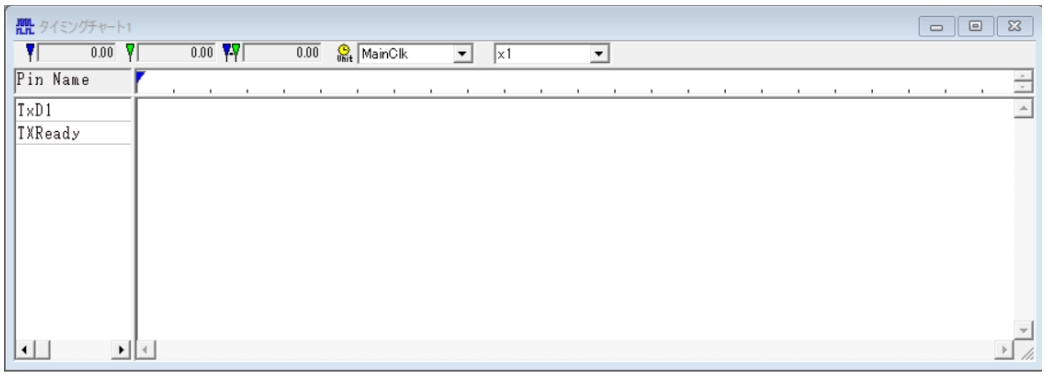
同様に「2」では、「P30/・・・・/SCL11」を選びます。



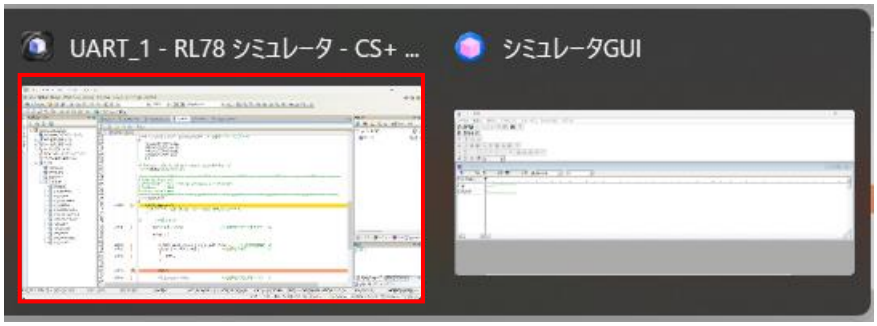
これで、接続する端子の指定が終わったので、最後に右側の「機能名」に適当な名前を入力します。ここで、入力した名前がタイミングチャートの「Pin Name」に表示されます。

ここでは、以下のように指定しました。何も設定しないと、接続端子名が表示されることになります。「OK」ボタンをクリックして設定を終了します。

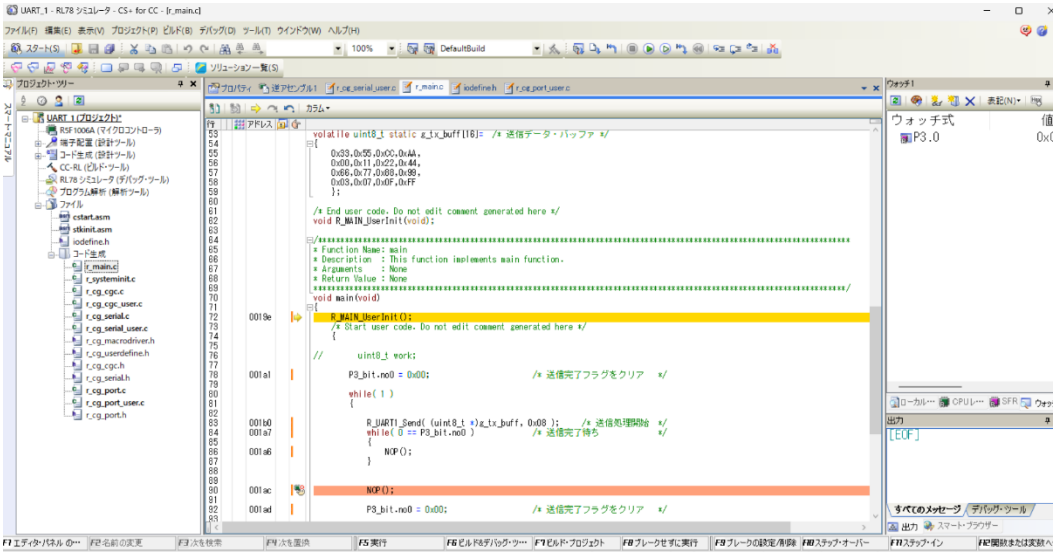
これで、以下のようになります。



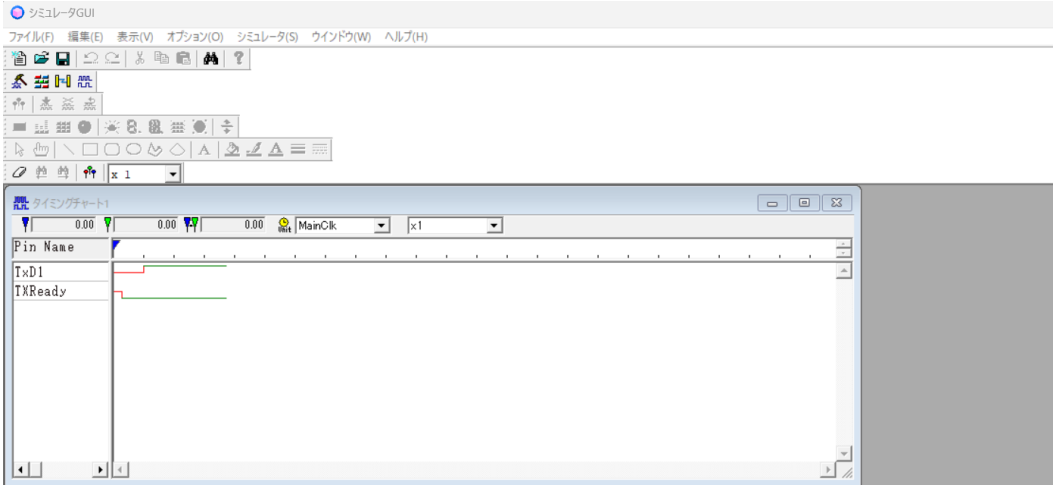
この状態で、一度シミュレータのウィンドウに戻り、シミュレータを終了します。



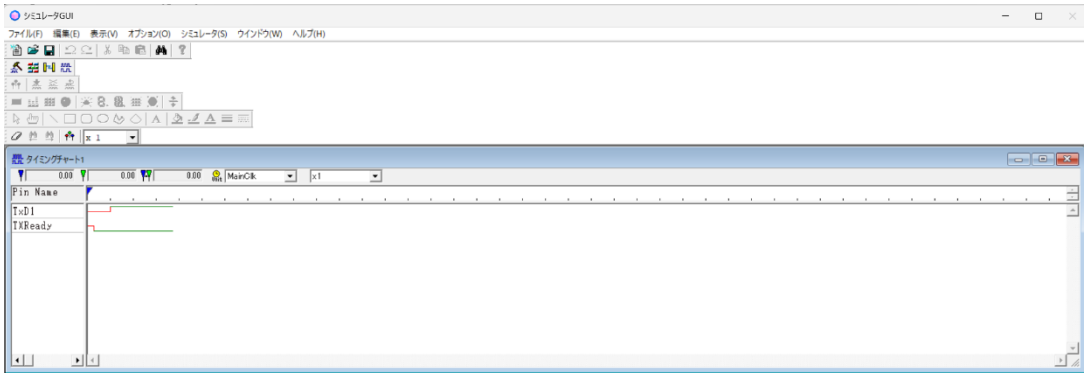
以降はシミュレータにビルド結果をダウンロードすると、以下のような画面となります。



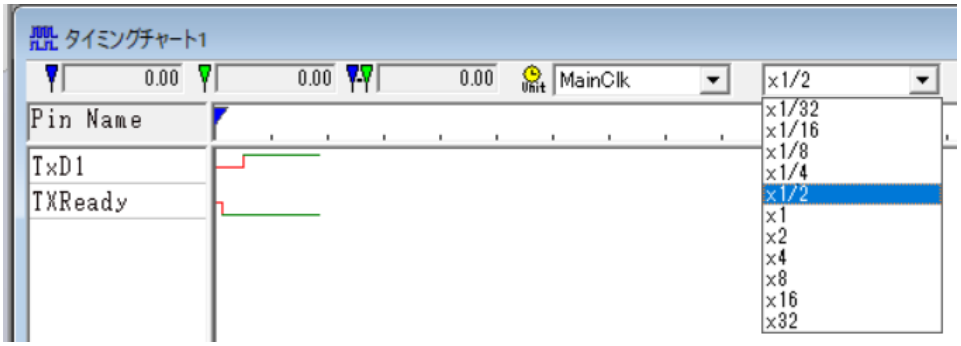
この状態で、「シミュレータ GUI」ウィンドウを選択すると、以下のように main の位置までの変化を見ることができます。



後は、必要に応じて、タイミングチャートの表示範囲を広げてください。



また、下に示すように、サンプリングする刻みを変更してみてください。



以上