Emulator上でのRISC-Vプロセッサによる Linuxブート

Linux Boot with RISC-V Processor on Cadence Emulator

2021/04/23

荒川 文男

Fumio Arakawa

東京大学 大学院 工学系研究科 システムデザイン研究センター

d.lab, The University of Tokyo

AI設計拠点のEmulatorの概要

- 1チップ大規模データをEDAシミュレーションより高速にエミュレーションできます。
- PCIe等のエミュレータ専用IPモデルの整備を進めております。
- シミュレータとエミュレータの協調設計検証が出来る専用サーバの準備を進めています。
  設計検証効率を向上させるエミュレータ活用方法を研究開発しています。
- 論理検証、システム検証、性能消費電力予測に適用できます。
- 1,200TBのストレージなど整備を進めています。
- セキュアにリソースを専有できます。
- リソース増強などプロジェクトニーズに対応できます。

AI設計拠点のEmulatorの仕様

- ASIC based Emulator、High Capacity and Optimized Verilog IF
- 72 Boards、576 Domains
- 最大ゲート規模：2,304MGate
- 最大ユーザメモリ規模：4,608GByte
- ライブラリ
  - ◆エミュレータ検証専用AXI, PCIe, HDMI, MIPI, USBのライブラリの整備を進めています。

cf.) https://ai-chip-design-center.org/aidc2020/aidc2020_wp_Public/public_tools/#Emulator

目次

- AI設計拠点のEmulatorの概要
- 東工大一色研のRISC-Vコアと検証環境
- Emulatorへのポーティング
  - Emulatorの制約と対応
  - UARTトランザクタの活用
  - 外部メモリの実装
  - UART速度の変更
- Emulator活用のノウハウ
  - ホストとEmulatorのインタフェース
  - 波形データ取得方法とデータ量削減方法
  - メモリデータ取得方法
  - simvisionでの信号追跡方法
- Emulation速度
  - パラメータの違い、波形取得の有無の影響
  - RTLとGate、EmulationとSimulationの比較

東工大一色研のRISC-Vコアと検証環境

- ５段パイプライン、命令・データキャッシュ、MMUあり
- 東工大一色研の高位システム設計検証環境で開発
  - C/C++からVerilog RTL、テストベンチを生成
- システム構成
  - RISC-Vコア、AXIバス、外部メモリI/F、UART I/F、SPI I/F
- 外部UARTモジュール
  - UART出力される文字コードを$write, $displayで表示
- テストベンチ
  - Verilogのinitial文で動作記述、クロック・リセットも記述
  - 毎サイクル、テスト対象の入出力ログを読み出し
  - 検証のために外部端子以外も出力し32ビットに圧縮してチェック
  - 入力ログはテスト対象に入力
  - ログはバークレーブートローダ（bbl）によるLinuxのブート

Emulatorへのポーティング

Emulatorの制約と対応

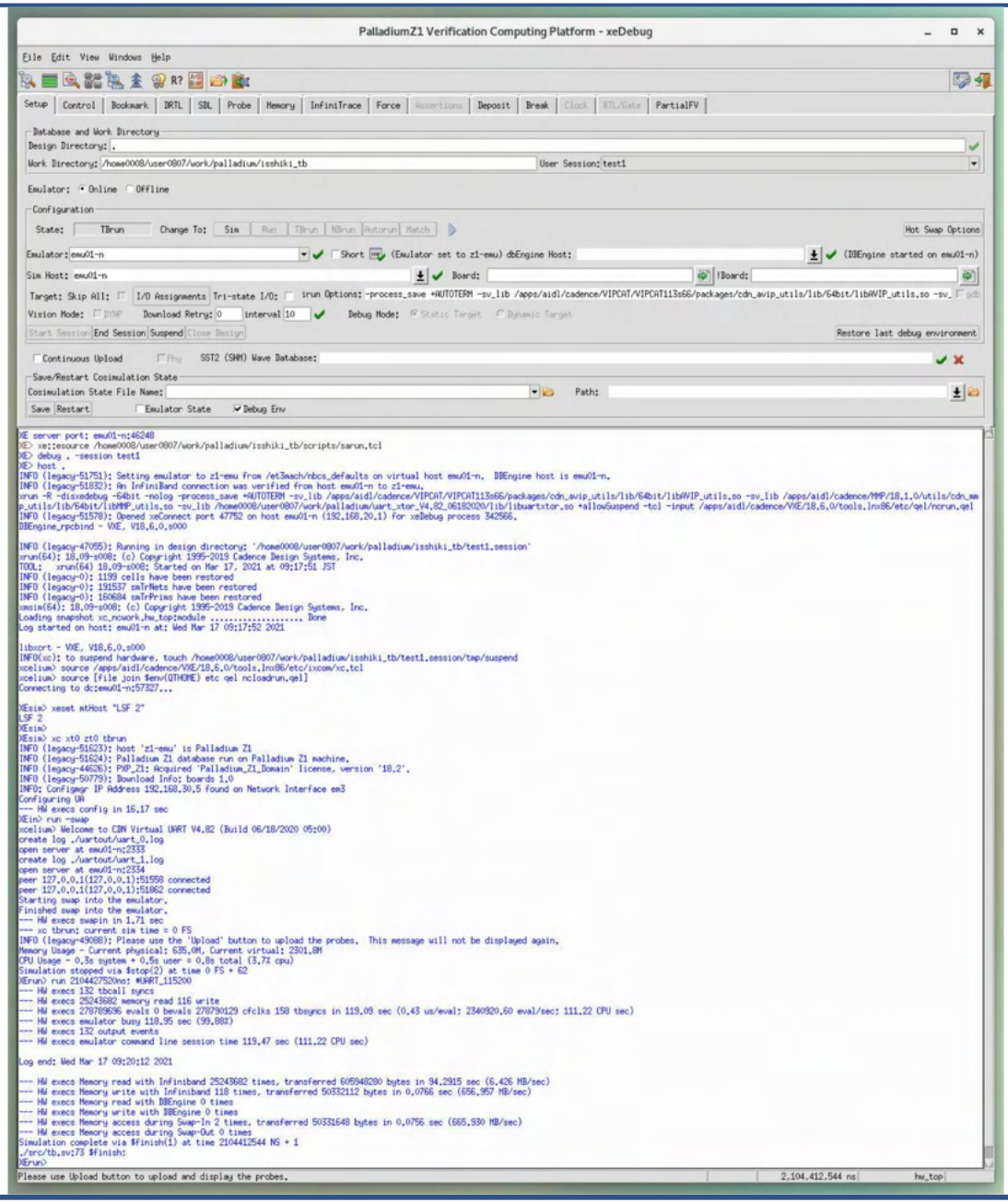
- Initial文による**動作記述**はEmulator HWに**マッピング不可**
  - ただし、初期化にはInitial文が使える
  - 合成可能な記述はマッピングできる
  - 動作記述の**テストベンチ**はEmulator用に**全面書換え**
- **100M要素を超える配列**定義はエラーとなる
  - Linuxのブートログは約116Mステップ
  - 2分割で対応し、配列初期化でログ読込み
  - ただし、宣言の後に /* sparse */ を付けると要素数$2^{31}-1$までOK
- **$write**, **$display**はワーニングを出して無視する
  - ただし、ixcomで**+iscdisp**オプションを付ければconsoleに出力できる
  - しかし、 Emulator制御の入出力と混ざってしまう
- AI設計拠点環境の制約
  - UARTトランザクタ使用時はホストにemu01-nを指定
  - com01-nでは動作しない

UARTトランザクタの活用

- UARTトランザクタはCadence社のAEWare *)
- 事前にJob投入端末でxhost +する
- 複数をインスタンス化することで、複数の端末ウインドウを開ける
- Linuxのブートでは、一つをバークレー・ブートローダー（bbl）出力に、もう一つをEmulation進行状況、ログとの比較結果表示に使用
- EmulatorのConsole出力と分離することで、 Emulator制御の入出力と混ざってしまうことを防げる
- Linuxホストのscreenやminicomと同様なインタフェース

*) AEWare：Cadence社製品の顧客に無料無保証で提供している
ソフトウェアやスクリプト、サンプル・デザインなど

EmulationのConsleウインドウ

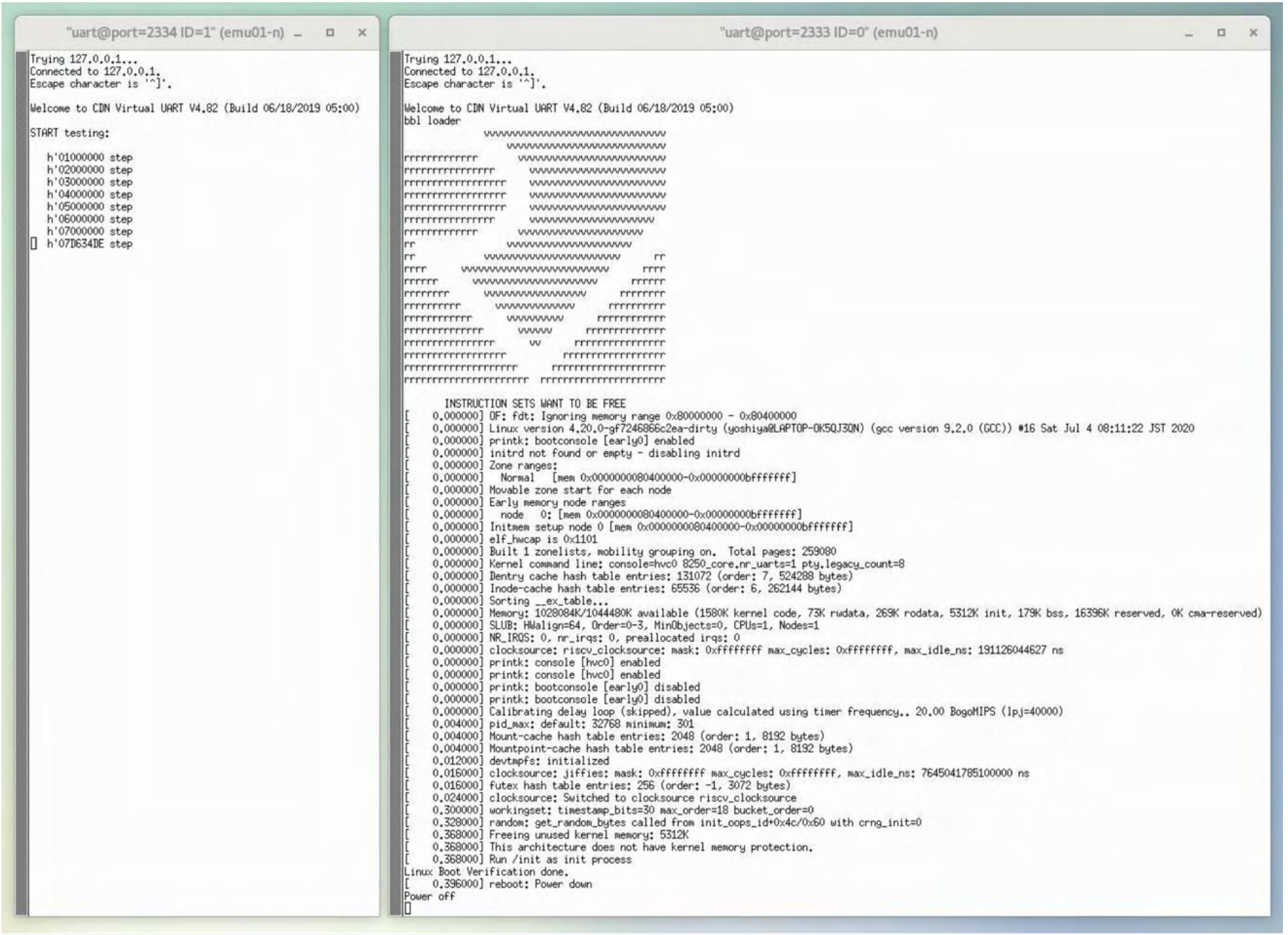


$write, $displayの場合のconsoleへの表示

```
[     317] <UART_AXI> (activate) : baud_period(     2), bit_count( 8), stop_count(1), parity_type(3)
[    11665] <UART_X.RX> msg = bbl loader
                                [    62604] <UART_X.RX> msg =            vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
        [    64594] <UART_X.RX> msg =               vvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
        [    66584] <UART_X.RX> msg = rrrrrrrrrrrrr      vvvvvvvvvvvvvvvvvvvvvvvvvvvv
        [    68552] <UART_X.RX> msg = rrrrrrrrrrrrrrr     vvvvvvvvvvvvvvvvvvvvvvvvvv
        [    70542] <UART_X.RX> msg = rrrrrrrrrrrrrrrrr    vvvvvvvvvvvvvvvvvvvvvvvvv
        [    72532] <UART_X.RX> msg = rrrrrrrrrrrrrrrrr    vvvvvvvvvvvvvvvvvvvvvvvvv
        [    74522] <UART_X.RX> msg = rrrrrrrrrrrrrrrrr    vvvvvvvvvvvvvvvvvvvvvvvvv
        [    76490] <UART_X.RX> msg = rrrrrrrrrrrrrrr     vvvvvvvvvvvvvvvvvvvvvv
        [    78480] <UART_X.RX> msg = rrrrrrrrrrrrr      vvvvvvvvvvvvvvvvvvvvvv
        [    80470] <UART_X.RX> msg = rr             vvvvvvvvvvvvvvvvvvvvvv
        [    82460] <UART_X.RX> msg = rr          vvvvvvvvvvvvvvvvvvvvvvvvvv      rr
        [    84428] <UART_X.RX> msg = rrrr      vvvvvvvvvvvvvvvvvvvvvvvvvvvv      rrrr
        [    86418] <UART_X.RX> msg = rrrrrr      vvvvvvvvvvvvvvvvvvvvvv      rrrrrr
        [    88408] <UART_X.RX> msg = rrrrrrrr      vvvvvvvvvvvvvvvvvv      rrrrrrrr
        [    90398] <UART_X.RX> msg = rrrrrrrrrr      vvvvvvvvvvvvvv      rrrrrrrrrr
        [    92366] <UART_X.RX> msg = rrrrrrrrrrrr      vvvvvvvvvv      rrrrrrrrrrrr
        [    94356] <UART_X.RX> msg = rrrrrrrrrrrrrr      vvvvvv      rrrrrrrrrrrrrr
        [    96346] <UART_X.RX> msg = rrrrrrrrrrrrrrrr      vv      rrrrrrrrrrrrrrrr
        [    98336] <UART_X.RX> msg = rrrrrrrrrrrrrrrrrr          rrrrrrrrrrrrrrrrrr
        [   100304] <UART_X.RX> msg = rrrrrrrrrrrrrrrrrrrr      rrrrrrrrrrrrrrrrrrrr
        [   102294] <UART_X.RX> msg = rrrrrrrrrrrrrrrrrrrrrr  rrrrrrrrrrrrrrrrrrrrrr
        [   102376] <UART_X.RX> msg =
                                    [   104079] <UART_X.RX> msg =       INSTRUCTION SETS WANT TO BE FREE
            [   594438] <UART_X.RX> msg = [    0.000000] OF: fdt: Ignoring memory range 0x80000000 -  0x80400000
                                [   610007] <UART_X.RX> msg = [    0.000000] Linux version 4.20.0-gf7246866c2ea-dirty (yo shiya@LAPTOP-OK5QJ3QN) (gcc version 9.2.0 (GCC)) #16 Sat Ju l 4 08:11:22 JST 2020
                        [   637816] <UART_X.RX> msg = [    0.000000] printk: bootconsole [early0] enabled
    [   672822] <UART_X.RX> msg = [    0.000000] initrd not found or empty - disabling initrd
                                    [   738145] <UART_X.RX> msg = [    0.000000] Zone ranges:
                  [   756258] <UART_X.RX> msg = [    0.000000]   Normal   [mem 0x0000000080400000-0x0000000 0bfffffff]
                                [   770942] <UART_X.RX> msg = [    0.000000] Movable zone start for each node
       [   782038] <UART_X.RX> msg = [    0.000000] Early memory node ranges
            [   798626] <UART_X.RX> msg = [    0.000000]   node   0: [mem 0x0000000080400000-0x000000 00bfffffff]
                                [   818448] <UART_X.RX> msg = [    0.000000] Initmem setup node 0 [mem 0x0000000080400000 -0x00000000bfffffff]
                        [  34612065] <UART_X.RX> msg = [    0.000000] elf_hwcap is 0x1101
                [  34671125] <UART_X.RX> msg = [    0.000000] Built 1 zonelists, mobility grouping on.  To tal pages: 259080
                                [  34691052] <UART_X.RX> msg = [    0.000000] Kernel command line: console=hvc0 8250_core. nr_uarts=1 pty.legacy_count=8
                    [  35248318] <UART_X.RX> msg = [    0.000000] Dentry cache hash table entries: 131072 (ord er: 7, 524288 bytes)
                            [  35524579] <UART_X.RX> msg = [    0.000000] Inode-cache hash table entries: 65536 (order : 6, 262144 bytes)
                              [  35538600] <UART_X.RX> msg = [    0.000000] Sorting __ex_table...
                [  54536675] <UART_X.RX> msg = [    0.000000] Memory: 1028084K/1044480K available (1580K k ernel code, 73K rwdata, 269K rodata, 5312K init, 179K bss,  16396K reserved, 0K cma-reserved)
                  [  54865390] <UART_X.RX> msg = [    0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, C PUs=1, Nodes=1
                            [  55022560] <UART_X.RX> msg = [    0.000000] NR_IRQS: 0, nr_irqs: 0, preallocated irqs: 0
                                    [  55068740] <UART_X.RX> msg = [    0.000000] clocksource: riscv_clocksource: mask: 0xffff ffff max_cycles: 0xffffffff, max_idle_ns: 191126044627 ns
 [  55104321] <UART_X.RX> msg = [    0.000000] printk: console [hvc0] enabled
          [  55112209] <UART_X.RX> msg = [    0.000000] printk: console [hvc0] enabled
          [  55127867] <UART_X.RX> msg = [    0.000000] printk: bootconsole [early0] disabled
     [  55136841] <UART_X.RX> msg = [    0.000000] printk: bootconsole [early0] disabled
     [  55178932] <UART_X.RX> msg = [    0.000000] Calibrating delay loop (skipped), value calc ulated using timer frequency.. 20.00 BogoMIPS (lpj=40000)
 [  55205392] <UART_X.RX> msg = [    0.000000] pid_max: default: 32768 minimum: 301
      [  55461385] <UART_X.RX> msg = [    0.000000] Mount-cache hash table entries: 2048 (order:  1, 8192 bytes)
                              [  55491671] <UART_X.RX> msg = [    0.000000] Mountpoint-cache hash table entries: 2048 (o rder: 1, 8192 bytes)
                          [  56618804] <UART_X.RX> msg = [    0.008000] devtmpfs: initialized
                [  56987512] <UART_X.RX> msg = [    0.008000] clocksource: jiffies: mask: 0xffffffff max_c ycles: 0xffffffff, max_idle_ns: 7645041785100000 ns
      [  57017806] <UART_X.RX> msg = [    0.008000] futex hash table entries: 256 (order: -1, 30 72 bytes)
                                  [  57822178] <UART_X.RX> msg = [    0.016000] clocksource: Switched to clocksource riscv_c locksource
                                [ 102291943] <UART_X.RX> msg = [    0.292000] workingset: timestamp_bits=30 max_order=18 b ucket_order=0
                              [ 106430213] <UART_X.RX> msg = [    0.316000] random: get_random_bytes called from init_oo ps_id+0x4c/0x60 with crng_init=0
                  [ 112406282] <UART_X.RX> msg = [    0.356000] Freeing unused kernel memory: 5312K
      [ 112423064] <UART_X.RX> msg = [    0.356000] This architecture does not have kernel memor y protection.
                                  [ 112438500] <UART_X.RX> msg = [    0.356000] Run /init as init process
             [ 114928452] <UART_X.RX> msg = Linux Boot Verification done.
```

表示が乱れる

UARTトランザクタの端末ウインドウ



外部メモリの実装

- 外部メモリからの入力がログ出力では、**サイクル数変化に対応できない**ので、外部メモリを実装した。
- Cadence社から、AXI、SDRAM、SDカードなどのインタフェースのメモリモジュールが提供されており、適宜、使用することができる。
- しかし、ログと同一動作タイミングのメモリはなかったため、自作した。
- ログでは、入力から1.5サイクルで出力を返している。
- アドレスマップでは、32ビットアドレス空間の半分が外部メモリであるが、配列要素数の上限を超える[1)]ので、**必要部分のみを実装**し、非実装領域にアクセスした場合は**エラーを報告して終了**するようにした。
- 何度か試行錯誤して実装領域を増やし、Linuxがブートするようになった。

1) 前述の /* sparse */ を使えば回避可能

UART速度の変更

- Cadence社のUARTトランザクタはUART信号のクロックと**その16倍のクロック**を入力して動作させる。
- 東工大一色研のRISC-Vコアの検証環境ではUART信号のクロックがメインクロックの1/2であるため、16倍のクロックは**メインクロックの8倍**速い。
- **Emulatorの速度は、最速クロックに強く依存する**ため、この状況ではUARTトランザクタがEmulatorの速度を決めてしまう。
- そこで、 UART信号のクロックをメインクロックの1/2から1/234にして、**最速がメインクロック**になるように変更した。
- 1/234は115,200bpsに相当し、1/2の次に速い設定である。
- この結果、 Emulatorの**速度は大幅に向上**した。
- 一方、 UART信号が大幅に遅くなったため、これがネックとなり、Linuxのブートは、約116Mサイクルから**約132Mサイクルに増加**した。

Emulator活用のノウハウ

よく参照するマニュアル

- docが沢山あって、参照すべきマニュアルが分かりにくい

- 実行関係では、VXEのdoc
  - 例えば　/apps/aidl/cadence/VXE/20.05.001.1/doc/
- その下の下記をよく参照する
  - vxecmdref/vxecmdref.pdf
  - vxeUserGuide/vxeUserGuide.pdf

- ただし、結構わかりにくいし、統一感がないので、先入観を捨てて読むことが重要
- 類似の記述からの類推は失敗のもと

ホストとEmulatorのインタフェース

- Task, functionによる制御の例
  - HW側のUARTに1文字（8 bit）出力するtaskをHost側で20回呼んで、文字列を表示

HW側

```
`ifdef IXCOM_COMPILE
initial begin $ixc_ctrl("tb_export", "sendData"); end
`endif
task sendData(input [7:0] data); begin
                                @(posedge clku); rx <= 0; //START bit
  for(int i=0;i<8;i++) begin @(posedge clku); rx <=data[i]; end
                                @(posedge clku); rx <= 1;      end
endtask
```

Host側

```
string CRLF = {8'h0d,8'h0a};
string msgi = {CRLF,"START testing:",CRLF,CRLF};
...
for (i = 0; i <  20; i = i + 1) hw_top.dut1.sendData(msgi[i]);
```

ホストとEmulatorのインタフェース

- export_read, export_eventの例
  - export_eventはevent listに使うときは必要らしい
  - とりあえず入れておけば、Errorの可能性は減る

HW側

```
`ifdef IXCOM_UXE
initial begin  $export_event(hw_top.dut1.abort);
               $export_read (hw_top.dut1.abort); end
`endif
```

Host側

```
 if(hw_top.dut1.abort) break;
```

  - abortがアサートされたら、ループを抜ける処理

波形データ取得方法とデータ量削減方法

- xeDebugの実行.tclでDatabaseの設定・無効化・有効化

```
set dbname <ディレクトリ名>
…
database -open    $dbname –cont
run <数値>ns
database -disable $dbname; # disable – enable 間は取得しない
run <数値>ns
database  -enable $dbname
run <数値>ns
```

- xeDebugの波形生成.scrで

```
set dbname <ディレクトリ名>
host  -offline $dbname/wave01
database -open $dbname/wave01_emu
probe -create hw_top -all -depth all; # all signals
または  probe -create -allnets -all -depth 1; # 1階層まで、など
database -upload
```

メモリデータ取得方法

- 波形データ取得方法ではメモリ（配列）データは取得できない
- runを止めて、memoryコマンドでファイルに出力

```
set <設定名> <出力ディレクトリ名>
…
run <数値>ns
memory -dump %readmemh <インスタンス名>.<配列名> ¥
                       -file $<設定名>/<出力ファイル名> ¥
                       -start <アドレス> -end <アドレス>
run <数値>ns
…
```

- 波形データのように値の時間変化を追える訳ではない
- 入出力信号の観測では不十分な時の最後の手段

simvisionでの信号追跡方法

- simvisionで波形データを読み込むだけでは信号追跡できない
- 波形表示起動コマンド例

```
setenv WAVE_NAME <ディレクトリ名>
simvision -64       -wave    ${WAVE_NAME}/wave01.shm ¥
                             ${WAVE_NAME}/wave01_emu.shm ¥
                    -input   ${WAVE_NAME}/wave01.svwf &
```

- 信号追跡時の起動コマンド例

```
setenv WAVE_NAME <ディレクトリ名>
simvision -64       -cdslib xc_work/cds.lib ¥
                    -snap xc_ncwork.hw_top:module ¥
                    ${WAVE_NAME}/wave01.shm ¥
                    ${WAVE_NAME}/wave01_emu.shm &
```

- 起動後、信号を選んでSend To Schematic Tracerで起動
  - ◆ クリックで入力側、出力側の双方にTraceできる
  - ◆ Unknown_<番号>の箱の中身は追跡できないが、
    Send To Source Browserで信号の記述を表示させることができる

Emulation速度

Emulation速度

- ポーティング当初は116Mサイクルの実行は数分程度であった。
- しかし、デバッグのために波形生成すると、実行に約10分、波形生成に2時間程度かかり、ファイルサイズは80GB程度になった。
- そこに、 UARTトランザクタを実装すると、実行に数時間かかり、波形生成は1日たっても終わらなくなった。
- そこで、体感速度をデータ化するため、パラメータを振って実行時間・CPU時間を収集した。
- 波形生成時にLinuxのブート終了まで計測するのは困難かつ非効率なので、1Mと10Mサイクル実行時の時間を収集した。
- Gate記述はRISC-Vコア部分をCadence社のGenusで合成置換したもの。
- Simulation時間は、EmulatorでHWをOFFした時の実行時間
- SimulationではLinuxブートに数百時間かかると推定されるため、1Mサイクル実行時の時間を収集し、外挿して求めた。
- RTLのEmulationで、最速クロック以外はEmulation速度に影響を与えなかったので、RTLとGate、 EmulationとSimulationの比較では、代表的な２組のパラメータ設定で実行した。

パラメータの違い、波形取得の有無の影響

	parameters				bootcycles	w/o wave		w/ wave						
								1M cycles			10M cycles			
						exec.	CPU	exec.	CPU	wave all	exec.	CPU	wave 1	wave all
1	log	-	Yes	1/2	116M	13'38"	718.98"	1'35"	5.79"	3'49"	8'36"	54.89"	7'12"	33'44"
2	memory module	Yes	Yes	1/2	116M	13'52"	730.56"	1'36"	6.25"	3'58"	8'33"	62.06"	8'16"	34'43"
3	memory module	No	Yes	1/2	116M	13'44"	718.33"	1'36"	6.28"	3'58"	7'48"	62.60"	7'28"	35'08"
4	memory module	Yes	No	1/2	116M	14'30"	785.66"	1'19"	6.44"	4'06"	7'30"	64.28"	6'54"	33'56"
5	memory module	No	No	1/2	116M	14'00"	754.86"	1'18"	6.23"	3'57"	8'10"	61.85"	6'48"	33'31"
6	memory module	Yes	No	1/234	132M	2'18"	107.82"	0'37"	0.89"	0'46"	1'32"	8.47"	1'27"	7'03"
7	memory module	No	No	1/234	132M	2'20"	110.47"	0'38"	0.88"	0'46"	1'29"	8.14"	1'27"	6'53"

- exec. time: "Log started on host:" to "Log end:"
- wave time: "Log end:" to "100% written"
- Parameters
  - external memory: "log" or "memory module"
  - log check every cycle, error check for every memory access
  - UART speed: 1/2 or 1/234 (115,200 bps)

パラメータの違い、波形取得の有無の影響

	parameters				bootcycles	w/o wave		w/ wave						
								1M cycles			10M cycles			
						exec.	CPU	exec.	CPU	wave all	exec.	CPU	wave 1	wave all
1	log	–	Yes	1/2	116M	13'38″	718.98″	1'35″	5.79″	3'49″	8'36″	54.89″	7'12″	33'44″
2	memory module	Yes	Yes	1/2	116M	13'52″	730.56″	1'36″	6.25″	3'58″	8'33″	62.06″	8'16″	34'43″
3	memory module	No	Yes	1/2	116M	13'44″	718.33″	1'36″	6.28″	3'58″	7'48″	62.60″	7'28″	35'08″
4	memory module	Yes	No	1/2	116M	14'30″	785.66″	1'19″	6.44″	4'06″	7'30″	64.28″	6'54″	33'56″
5	memory module	No	No	1/2	116M	14'00″	754.86″	1'18″	6.23″	3'57″	8'10″	61.85″	6'48″	33'31″
6	memory module	Yes	No	1/234	132M	2'18″	107.82″	0'37″	0.89″	0'46″	1'32″	8.47″	1'27″	7'03″
7	memory module	No	No	1/234	132M	2'20″	110.47″	0'38″	0.88″	0'46″	1'29″	8.14″	1'27″	6'53″

- Emulatorの速度は、**最速クロックに強く依存**する。
- 実行時間は、パラメータの違いによる差より**ばらつきの方が大きい**。
- **波形生成の区間、depth**は実行時間に大きく影響する。
- CPU時間は、波形生成の有無による影響が小さい。
- 実行速度はCPU時間で**約1.2M/s**、実行時間で0.9～1.1M/s

RTLとGate、EmulationとSimulationの比較

	parameters			boot cycles		Linux Boot			1M cycles		
	ext mem.	log check	UART			RTL	Gate	G/R	RTL	Gate	G/R
1	log	Yes	1/2	116M	Emu.	13'38"	15'06"	1.11	0'46"	2'01"	2.63
					Sim.	*262:46'20"*	*561:47'40"*	*2.14*	2:15'55"	4:50'35"	2.14
					S/E	*1,156*	*2,232*		177	144	
7	mem.	No	1/234	132M	Emu.	2'20"	2'31"	1.08	0'22"	0'53"	2.41
					Sim.	*145:09'48"*	*609:19'36"*	4.20	1:05'59"	4:36'58"	4.20
					S/E	*3,733*	*14,527*		180	314	

◆ *Italic numbers are estimated values with extrapolation of 1M-cycle executions.*

- 1M cyclesのEmulationでは初期化時間が無視できず、Linux Bootの1/100未満の実行に1/18～1/3の時間がかかる。したがって、**1M cyclesでの比較は不適切**。
- 一方、 Simulationでは初期化時間が無視でき、**外挿によるLinux Boot時間推定**は正しいと期待できる。
- UART=1/2設定の影響は変化する。Emulationでは大、SimulationのRTLでは小、Gateではほぼ無し。変化の少ない**UART=1/234での比較**が妥当。
- 赤字部分がこれに相当し、**RTLで3,700倍、Gateで14,500倍高速化**。

まとめ

- Cadence社のEmulator上でRISC-Vコアによる Linuxのブートに成功
- RISC-Vコアは東工大一色研の高位システム設計検証環境で開発したコア
- Emulation環境構築では、UARTトランザクタを活用して、Linux端末ウインドウおよびEmulation状況のモニタリングを実現
- 約132M cycleの実行時間は最速で2分20秒程度
- 速度はCPU時間で約1.2M cycles/s、実行時間で0.9～1.1M cycles/s

謝辞

本成果は、国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）の委託業務「AIチップ開発加速のためのイノベーション推進事業、研究開発項目②：AIチップ開発を加速する共通基盤技術の開発」の結果得られたものです。また，本研究に用いたRISC-Vコア及び周辺論理は東京工業大学一色研究室より提供していただいたたものです。

ご清聴有難うございました