*hyper*stone electronics tutorial

E1-32/E1-16
and
Software Development Tools

hyperstone electronics

Table Of Contents (1)

Unifying RISC and DSP

hyperstone electronics

Table Of Contents (2)

Unifying RISC and DSP

hyperstone electronics

Table Of Contents (3)

Unifying RISC and DSP

hyperstone electronics

Table Of Contents (4)

Unifying RISC and DSP

hyperstone electronics

Table Of Contents (5)

Unifying RISC and DSP

hyperstone electronics

Memory

Unifying RISC and DSP

- **Memory**
  - Memory Address Space
  - Memory Control Register
  - Memory Write Access Modes
  - Bus Control Register
  - Connecting DRAM
  - Connecting Boot EPROM
  - Internal RAM

Memory Address Space

q Connecting External Memory

- 32-bit (E1-16: 16-bit) wide data bus
- 26-bit (E1-16: 22-bit) wide address bus
- memory address space of 4 GByte in total
- memory address space is divided into five memory areas
- each memory area with separate bus timing and bus width

Memory Area	Memory Address Range	Data Bus Width	Memory Type
MEM0	$0000\ 0000_{16}$..$3FFF\ FFFF_{16}$	32, 16, 8	ROM, SRAM, DRAM
MEM1	$4000\ 0000_{16}$..$7FFF\ FFFF_{16}$	32, 16, 8	ROM, SRAM
MEM2	$8000\ 0000_{16}$..$BFFF\ FFFF_{16}$	32, 16, 8	ROM, SRAM
IRAM	$C000\ 0000_{16}$..$DFFF\ FFFF_{16}$	32	internal RAM (on-chip)
MEM3	$E000\ 0000_{16}$..$FFFF\ FFFF_{16}$	32, 16, 8	ROM, SRAM

hyperstone electronics

Memory Control Register

Unifying RISC and DSP

- Memory Control Register MCR
  - memory types of MEM0
    - Fast Page Mode (FPM) DRAM
    - Extended Data Output (EDO) DRAM (E1-32X)
    - non-DRAM
  - individual parameters for MEM0..MEM3:
    - data bus width (32-bit, 16-bit, 8-bit)
    - memory bus hold cycles (skipping, inserting)
    - memory write access modes (E1-32X)
      - byte write strobe
      - byte enable signal
  - 32-bit write-only register
  - all bits set to one on Reset

9

Bit 21 of the MCR specify the type of memory connected to the memory area MEM0:

MCR(21) = 0 for DRAM
MCR(21) = 1 for non-DRAM

Bit 15 of the MCR specify the type of DRAM connected to the memory area MEM0:

MCR(15) = 0 for EDO DRAMs
MCR(15) = 1 for FPM DRAMs

Bits 7..0 of the write-only memory control register MCR defines the data bus width (32-bit, 16-bit, 8-bit) for MEM0..MEM3.

Bits 11..8 of the MCR specify a memory bus hold break for memory area MEM3..MEM0 respectively. The default setting is disabled. When memory bus hold break is enabled, bus hold cycles are skipped when the next memory access addresses the same memory area.

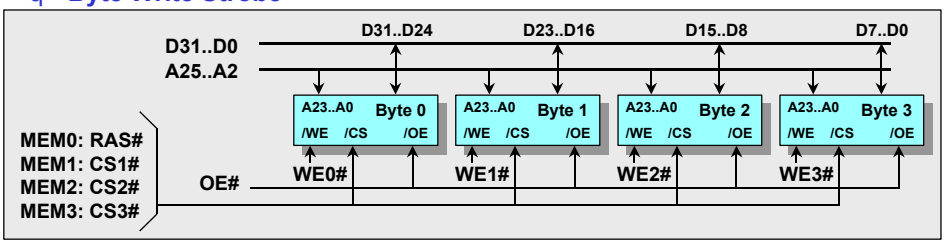
The number of inserted memory bus hold cycles can be specified in the bus control register BCR.

hyperstone electronics

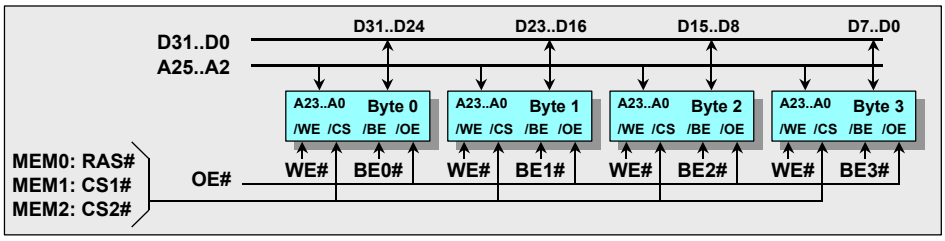
MCR: Memory Write Access Modes (E1-32X)

Unifying RISC and DSP

- Byte Write Strobe



- Byte Enable Signal



hyperstone electronics

Bus Control Register

Unifying RISC and DSP

- **Bus Control Register BCR**
  - **individual parameters for MEM0..MEM3:**
    - **address bus timing**
    - **data bus timing**
    - **DRAM page size**
    - **DRAM refresh rate**
    - **parity generation and checking**
  - **32-bit write-only register**
  - **all bits are set to one on Reset**

11

The write-only bus control register BCR defines the parameters (memory bus timing, DRAM page size, DRAM refresh rate, parity generation and checking) for accessing external memories located in address spaces MEM0..MEM3.

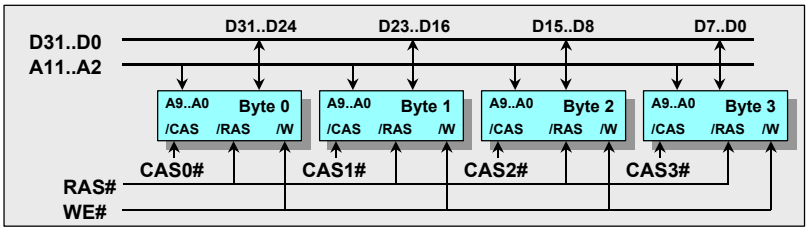
All bits of the MCR and the BCR are set to one on Reset and have to be initialized according to the external connected memories after Reset. These default settings represent the slowest memory bus timing. Thus all types of memories can operate with this moderate bus timing.

hyperstone electronics

Connecting DRAM, 32-bit Data Bus (only MEM0)

Unifying RISC and DSP

- **4MByte = 4 * 1Mx8 DRAM**



Page Size Code

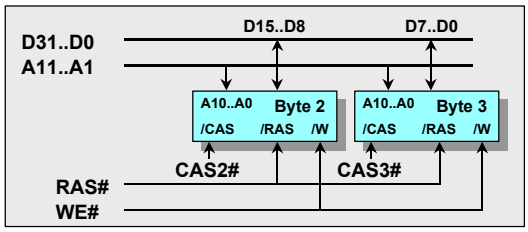
	Column Address Range		
BCR(6..4)	32-bit Bus Size	16-bit Bus Size	8-bit Bus Size
000	A15..A2	A15..A1	A15..A0
001	A14..A2	A14..A1	A14..A0
010	A13..A2	A13..A1	A13..A0
011	A12..A2	A12..A1	A12..A0
100	A11..A2	A11..A1	A11..A0
101	A10..A2	A10..A1	A10..A0
110	A9..A2	A9..A1	A9..A0
111	A8..A2	A8..A1	A8..A0

hyperstone electronics

Connecting DRAM, 16-bit Data Bus (only MEM0)

Unifying RISC and DSP

q **8MByte = 2 * 4Mx8 DRAM**



Page Size Code

	Column Address Range		
BCR(6..4)	32-bit Bus Size	16-bit Bus Size	8-bit Bus Size
000	A15..A2	A15..A1	A15..A0
001	A14..A2	A14..A1	A14..A0
010	A13..A2	A13..A1	A13..A0
011	A12..A2	A12..A1	A12..A0
100	A11..A2	A11..A1	A11..A0
101	A10..A2	A10..A1	A10..A0
110	A9..A2	A9..A1	A9..A0
111	A8..A2	A8..A1	A8..A0

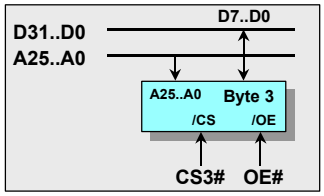
hyperstone electronics

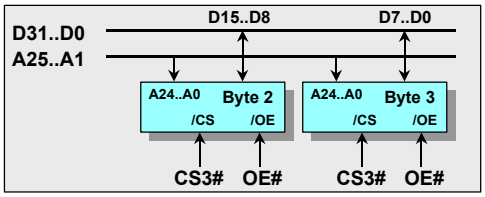
Connecting Boot EPROM (only MEM3)

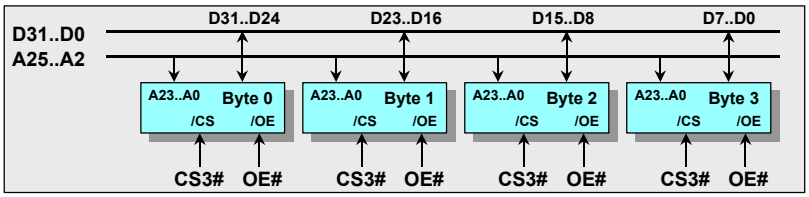
Unifying RISC and DSP

q Selecting Data Bus Width for MEM3

Input BOOTW	Input BOOTB	Data Bus Width
Don't care	HIGH	8-bit
LOW	LOW	16-bit
HIGH	LOW	32-bit







14

hyperstone electronics

Internal RAM

Unifying RISC and DSP

q **Internal RAM (IRAM)**

- 8 KBytes (E1-32: 4 KBytes) on-chip memory
- mapped to memory base address C000 0000_{16}
- wraps around modulo 8KBytes up to memory addess DFFF $FFFF_{16}$
- implemented as dynamic memory, needing refresh
- refresh rate is specified in bits 18..16 of Memory Control Register MCR (default is refresh disabled)
- one clock cycle access time
- automatic insertion of one wait cycles, if the target register of the load is addressed before the data is loaded into the target register:

```
MOVI     L0, $C0000000   ; first address in IRAM
LDW.R    L0, L1          ; LOAD word from address $C0000000 into L1
                         ; automatic insertion of one wait cycle
                         ; between LOAD and USE
ADDI     L1, 1           ; USE target register L1 of preceding load
```

15

8 KBytes (E1-32: 4 KBytes) of memory are provided on-chip. This internal RAM (IRAM) is mapped to the memory address C000 0000_{16} and wraps around modulo 8KBytes up to memory address DFFF $FFFF_{16}$. The IRAM is implemented as dynamic memory, needing refresh.

The refresh rate must be specified in the MCR bits 18..16 before any use. The number given in MCR(18..16) specifies the refresh rate in CPU clock cycles; e.g. 128 specifies a refresh cycle automatically inserted every 128 clock cycles. Each refresh cycle refreshes 16 bytes, thus, 256 refresh cycles are required to refresh the whole IRAM. Without refresh the dynamic cell can hold the data for about 80 ms. A high refresh rate does not degrade performance since the refresh cycles are inserted on idle IRAM cycles whenever possible.

In order to parallelize accesses to the internal RAM and the externel memory, a separate memory pipeline has been added for accesses to the IRAM. This means e.g. that a new instruction can be fetched from the IRAM while a data load or data store to external memory is still in progress.

The minimum delay for a load access is one cycle; that is, the data is not available in the cycle after the load instruction. One wait cycle is automatically inserted if the target register of the load is addressed before the data is loaded into the target register.

hyperstone electronics

Peripherals

Unifying RISC and DSP

- Peripherals
  - I/O Bus Access
  - UART of hyICE
  - I/O Address Modes
  - I/O Access with C Run-Time Library

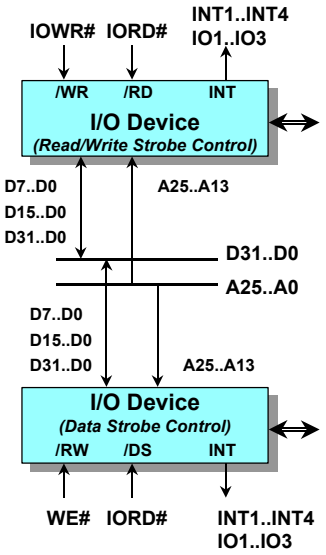
16

hyperstone electronics

I/O Bus Access

Unifying RISC and DSP

- I/O Address Bits 25..13 can be used as I/O Address
- I/O Bus Timing for an I/O Access is specified by bits 9..3 of the I/O Address
  - address setup time (bit 9, 8)
  - access time for read or write access (bit 7..5)
  - bus hold time after read or write access (bit 4, 3)
- Bit 10 of the I/O Address controls Device Mode
  - address bit 10 = 0
    - IORD# (I/O read strobe)
    - IOWR# (I/O write strobe)
  - address bit 10 = 1
    - IORD# (data strobe)
    - WE# (read/write direction)
- Bit 12 reserved for System Peripherals
- Data Bus Access with fixed 32-bit wide

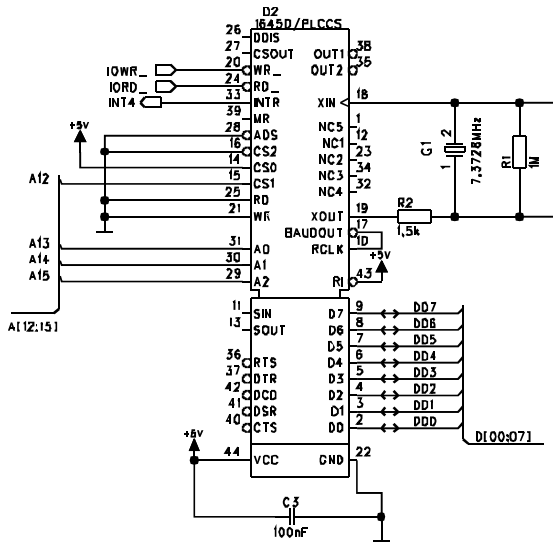


17

hyperstone electronics

I/O Bus Access: UART of hyICE (1)

Unifying RISC and DSP



- **UART Startech ST16450**
  - **I/O chip select with A12**
  - **8 Registers selected by A13..A15**
  - **IORD# (read strobe)**
  - **IOWR# (write strobe)**
  - **interrupt output to INT4**
  - **data bus D7..D0**

18

An own application should not use INT4, since this signal is used by the hyICE and the real-time operating system hyRTK.

hyperstone electronics

I/O Bus Access: UART of hyICE (2)

Unifying RISC and DSP

- **I/O Bus Timing, Device Control Mode and Register Address of UART**

```
SystemChipSelect   EQU  (  %1 << 12) ; Bit 12
DeviceControlMode  EQU  (  %0 << 10) ; Bit 10
AddressSetupTime   EQU  ( %11 <<  8) ; Bit 9, 8
AccessTime         EQU  (%101 <<  5) ; Bit 7, 6, 5
BusHoldTime        EQU  ( %11 <<  3) ; Bit 4, 3
UARTBaseAddress    EQU  SystemChipSelect+DeviceControlMode+ \
                        AddressSetupTime+AccessTime+BusHoldTime
UARTRegisterOffset EQU  (1<<13)      ; Bit 15, 14, 13
UARTRegister0      EQU  UARTBaseAddress+(UARTRegisterOffset * 0)
UARTRegister1      EQU  UARTBaseAddress+(UARTRegisterOffset * 1)
UARTRegister2      EQU  UARTBaseAddress+(UARTRegisterOffset * 2)
UARTRegister3      EQU  UARTBaseAddress+(UARTRegisterOffset * 3)
UARTRegister4      EQU  UARTBaseAddress+(UARTRegisterOffset * 4)
UARTRegister5      EQU  UARTBaseAddress+(UARTRegisterOffset * 5)
UARTRegister6      EQU  UARTBaseAddress+(UARTRegisterOffset * 6)
UARTRegister7      EQU  UARTBaseAddress+(UARTRegisterOffset * 7)
```

19

The assembler directive **`EQU`** is used to give constant expressions or string patterns a symbolic name. Any identifier used to define an equate must not have been previously defined.

Binary numbers are unsigned 32-bit integers beginning with the **`%`** character and followed by a sequence of the characters **`0`** or **`1`** with no spaces in between.

The assembler operator **`<<`** shifts an operand left by a number of bit positions.

hyperstone electronics

I/O Absolute Address Mode

Unifying RISC and DSP

q **I/O Absolute Address Mode**

Notation load instruction: `LDx.IOA 0, Rs, dis`

Notation store instruction: `STx.IOA 0, Rs, dis`

Data Type x is with: `W`: word; `D`: double-word;

- **Absolute addressing of peripheral devices**

```
LDW.IOA  0, L1, UARTRegister0  ; load word from UART reg. 0 to L1
LDW.IOA  0, L1, UARTRegister1  ; load word from UART reg. 1 to L1
LDW.IOA  0, L1, UARTRegister2  ; load word from UART reg. 2 to L1
```

20

I/O Absolute Address Mode:

Notation load instruction: **`LDx.IOA`** `0, Rs, dis`

Notation store instruction: **`STx.IOA`** `0, Rs, dis`

Data Type `x` is with:

`W`: word; **`D`**: double-word;

The displacement `dis` is used as an address into I/O address space.

Address bits one and zero of `dis` are treated as zero.

Execution of a memory instruction with I/O address mode does not disrupt any page mode sequence.

The I/O absolute address mode provides code efficient absolute addressing of peripheral devices and allows simple decoding of I/O addresses.

When on a load instruction only a byte or a halfword is placed on the lower part of the data bus, the higher-order bits are undefined and must be masked out before the loaded operand is used further.

hyperstone electronics

I/O Displacement Address Mode

Unifying RISC and DSP

q **I/O Displacement Address Mode**

Notation load instruction: `LDx.IOD Rd, Rs, dis`

Notation store instruction: `STx.IOD Rd, Rs, dis`

Data Type x is with: `W`: word; `D`: double-word;

- **Dynamic addressing of peripheral devices**

```
MOVI     L0, UARTRegister0      ; L0 = address of UART reg. 0
LDW.IOD  L0, L1, 0              ; load word from UART reg. 0 to L1
ADDI     L0, UARTRegisterOffset ; L0 = L0 + offset to next UART reg
LDW.IOD  L0, L1, 0              ; load word from UART reg. 1 to L1
ADDI     L0, UARTRegisterOffset ; L0 = L0 + offset to next UART reg
LDW.IOD  L0, L1, 0              ; load word from UART reg. 2 to L1
```

21

I/O Displacement Address Mode:

Notation load instruction: **`LDx.IOD`** `Rd, Rs, dis`

Notation store instruction: **`STx.IOD`** `Rd, Rs, dis`

Data Type `x` is with:

`W`: word; **`D`**: double-word;

The sum of the contents of the destination register `Rd` plus a signed displacement `dis` is used as an address into I/O address space.

The destination register `Rd` may denote any register **except** the status register SR.

Address bits one and zero of `dis` are treated as zero for the calculation of `Rd` + `dis`.

Execution of a memory instruction with I/O displacement address mode does not disrupt any page mode sequence.

The I/O displacement address mode provides dynamic addressing of peripheral devices.

When on a load instruction only a byte or halfword is placed on the lower part of the data bus, the higher-order bits are undefined and must be masked out before the loaded operand is used further.

C Run-Time Library: inpw()

- **inpw**

  Synopsis

  ```
  #include <io.h>
  unsigned long int inpw(unsigned long int portid);
  ```

  Description

  The macro `inpw` reads a 32-bit value from the address `portid` located in the *hyper*stone I/O address space and returns this value as `unsigned long int`.

  Returns

  The macro `inpw` returns the value read from I/O address space.

C Run-Time Library: outpw()

- **outpw**

  Synopsis

```
#include <io.h>
unsigned long int outpw(unsigned int long portid,
                        unsigned int long value);
```

  Description

  The macro `outpw` writes the 32-bit `value` to the address `portid` located in the *hyper*stone I/O address space.

  Returns

  The macro `outpw` returns the 32-bit value written to `portid`.

hyperstone electronics

Inputs and Outputs

Unifying RISC and DSP

- Inputs and Outpus
  - Function Control Register
  - Interrupt Inputs
  - Input Status Register
  - Wait Pin INT3
  - Outputs
  - Clock Output

Interrupts and Function Control Register

Unifying RISC and DSP

- **Function Control Register FCR**
  - **controls interrupt mask and polarity of**
    interrupt pins INT4..INT1 (interrupt inputs)
  - **controls interrupt mask, polarity and direction of**
    I/O pins IO3..IO1 (general inputs, general outputs or interrupt inputs)
  - **controls interrupt mask and priority of**
    internal timer interrupt
  - **controls polarity and behaviour of**
    clock output pin CLKOUT (only E1-32X)
  - **32-bit write-only register**
  - **all bits are set to one on Reset**

25

The write-only function control register FCR controls the polarity and interrupt mask of the interrupt pins INT4..INT1 and the I/O pins IO3..IO1, the timer interrupt mask and the priority of the internal timer interrupt.

Each of the four interrupt pins INT4..INT1 can cause a processor interrupt, when the corresponding interrupt mask bit INT4Mask..INT1Mask is cleared (bit 31, 30, 29 and 28). The E1-32/E1-16 only supports level-sensitive interrupts.

The corresponding polarity bit INT4Polarity..INT1Polarity (bit 27, 26, 25 and 24) determines whether the signal at the interrupt pin INT4..INT1 must be low (INTxPolarity = 0) or high (INTxPolarity = 1) to cause an interrupt.

The corresponding direction bit IO3Direction..IO1Direction (bit 10, 6 and 2) determines whether the I/O pins IO3..IO1 can be either used as general input or interrupt input (IOxDirection = 1) or as general output (IOxDirection = 0).

The corresponding polarity bit IO3Polarity..IO1Polarity (bit 9, 5 and 1) determines whether the signal at the I/O pin must be low (IOxPolarity = 0) or high (IOxPolarity = 1) to cause an interrupt, if used as interrupt input.

Each of the three pins IO3..IO1 can cause a processor interrupt, when the corresponding interrupt mask bit IO3Mask..IO1Mask is cleared (bit 8, 4, and 0) and the corresponding direction bit is set (IOxDirection = 1).

Bit 23 of the FCR enables or disables the internal timer interrupt. Bit 21..20 specify the priority of the timer interrupt. Priority 12, 10, 8 and 6 are selectable.

FCR: Interrupt Inputs INT4..INT1, IO3..IO1

Unifying RISC and DSP

```
INT4Mask          EQU   ( %0 << 31) ; Bit 31           enable INT4 interrupt
INT3Mask          EQU   ( %0 << 30) ; Bit 30           enable INT3 interrupt
INT2Mask          EQU   ( %0 << 29) ; Bit 29           enable INT2 interrupt
INT1Mask          EQU   ( %0 << 28) ; Bit 28           enable INT1 interrupt
INT4Polarity      EQU   ( %1 << 27) ; Bit 27           INT4 interrupt on high level
INT3Polarity      EQU   ( %1 << 26) ; Bit 26           INT3 interrupt on high level
INT2Polarity      EQU   ( %0 << 25) ; Bit 25           INT2 interrupt on low level
INT1Polarity      EQU   ( %0 << 24) ; Bit 24           INT1 interrupt on low level
IO3Direction      EQU   ( %1 << 10) ; Bit 10           IO3 input
IO2Direction      EQU   ( %1 <<  6) ; Bit  6           IO2 input
IO1Direction      EQU   ( %1 <<  2) ; Bit  2           IO1 input
IO3Mask           EQU   ( %0 <<  8) ; Bit  8           enable IO3 interrupt
IO2Mask           EQU   ( %0 <<  4) ; Bit  4           enable IO2 interrupt
IO1Mask           EQU   ( %0 <<  0) ; Bit  0           enable IO1 interrupt
IO3Polarity       EQU   ( %1 <<  9) ; Bit  9           IO3 interrupt on high level
IO2Polarity       EQU   ( %1 <<  5) ; Bit  5           IO2 interrupt on high level
IO1Polarity       EQU   ( %1 <<  1) ; Bit  1           IO1 interrupt on high level
IO3Control        EQU   (%11 << 12) ; Bit 13, 12       IO3 standard mode
TimerMask         EQU   ( %0 << 23) ; Bit 23           enable internal timer interrupt
TimerPriority     EQU   (%00 << 20) ; Bit 21, 20       Priority 12

FCRValue          EQU   INT4Mask     + INT3Mask      + INT2Mask     + INT1Mask     + \
                        INT4Polarity + INT3Polarity  + INT2Polarity + INT1Polarity + \
                        IO3Mask      + IO2Mask       + IO1Mask                      + \
                        IO3Polarity  + IO2Polarity   + IO1Polarity                  + \
                        IO3Direction + IO2Direction  + IO1Direction + IO3Control    + \
                        TimerMask    + TimerPriority
```

26

All bits of the function control register FCR are set to one on Reset. They have to be initialized according to the hardware environmnet and the desired function. The reserved bits 22, 19..18, 15..14, 11, 7 and 3 must not be changed when the FCR is updated.

A signal of a specified level on any of the interrupt request pins INT4..INT1 or on any of the general input-output pins IO3..IO1 (when configured as interrupt input) causes an interrupt exception when the interrupt-lock flag L is zero and the corresponding INTxMask or IOxMask bit in the FCR is not set.

Bit 15 of the status register SR is the interrupt-lock flag L. When the L flag is one, all Interrupt, Parity Error and Extended Overflow exceptions are inhibited. The interrupt-lock flag L is set by any exception (Reset, Interrupt etc.), thus no further interrupts can occur until the L flag is cleared.

Interrupt signals on INT4..INT1 and IO3..IO1 may be signaled asynchronously to the processor clock, they are not stored internally. A transition of INT4..INT1 or IO3..IO1 is effective after a minimum of three clock cycles. The response time may be much higher depending on the number of cycles to the end of the current instruction or the number of cycles until the interrupt-lock flag is cleared.

hyperstone electronics

Input Status Register

Unifying RISC and DSP

q **Input Status Register ISR**

- **Bits 6..4 reflects input level at pins IO3..IO1**
- **Bits 3..0 reflects input level at pins INT4..INT1**
- **input levels are not affected by the polarity bits in FCR**
- **input levels reflect always true signal at corresponding pin**
- **1 signals high level at input level**
- **32-bit read-only register**

27

The read-only input status register ISR reflects the input level at the pins IO3..IO1 as well as the input levels at the interrupt pins INT4..INT1.

The input levels are not affected by the polarity bits in the FCR register, they reflect always the true signal at the corresponding pins with a latency of 2..3 clock cycles, a “1” signals high level.

Bits 6..4 reflects the input level at the pins IO3..IO1.

The signal level of INT4..INT1 can be inspected in bit 3..0 of the ISR. Thus, with the corresponding INTxMask bit set, INT4..INT1 can be used just as input signals.

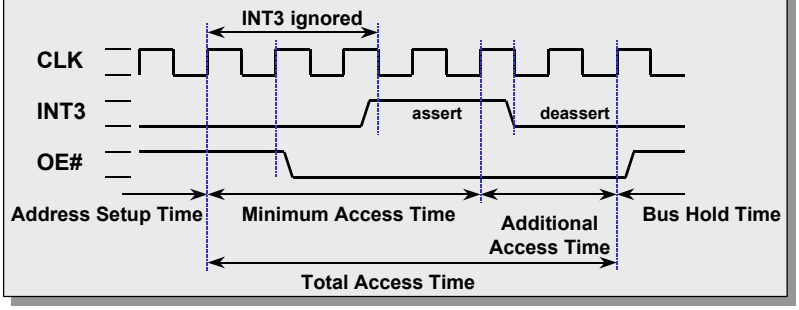
hyperstone electronics

Access with Wait Pin INT3 (E1-32X)

Unifying RISC and DSP

- Stretched I/O access
  Bit 11 of the I/O address enables wait-pin INT3 controlled I/O access
- Stretched memory access for memory area MEM2 and MEM3
  Bit 27 and 26 of the MCR enables wait-pin INT3 controlled memory access
- wait-pin INT3 controlles termination of access
- minimum access time of 4 cycles

Example: memory read access



28

hyperstone electronics

FCR: Outputs IO3..IO1

Unifying RISC and DSP

```
...
IO3Direction  EQU   ( %1 << 10) ; Bit 10          IO3 output
IO2Direction  EQU   ( %1 <<  6) ; Bit  6          IO2 output
IO1Direction  EQU   ( %1 <<  2) ; Bit  2          IO1 output
IO3Mask       EQU   ( %1 <<  8) ; Bit  8          IO3 Output reflects IO3Polarity
IO2Mask       EQU   ( %1 <<  4) ; Bit  4          IO2 Output reflects IO2Polarity
IO1Mask       EQU   ( %1 <<  0) ; Bit  0          IO1 Output reflects IO1Polarity
IO3Polarity   EQU   ( %1 <<  9) ; Bit  9          IO3 Polarity non-inverted
IO2Polarity   EQU   ( %1 <<  5) ; Bit  5          IO2 Polarity non-inverted
IO1Polarity   EQU   ( %0 <<  1) ; Bit  1          IO1 Polarity inverted
IO3Control    EQU   (%11 << 12) ; Bit 13, 12      IO3 standard mode
...

FCRValue      EQU   INT4Mask     + INT3Mask      + INT2Mask     + INT1Mask     + \
                    INT4Polarity + INT3Polarity  + INT2Polarity + INT1Polarity + \
                    IO3Mask      + IO2Mask       + IO1Mask                      + \
                    IO3Polarity  + IO2Polarity   + IO1Polarity                  + \
                    IO3Direction + IO2Direction  + IO1Direction + IO3Control    + \
                    TimerMask    + TimerPriority
```

*hyper*stone E1-32 Development Board with two on-board LED s

IO1 drives red LED1

IO2 drives green LED2

29

The I/O pins IO3..IO1 are configured as output if the corresponding direction bit IO3Direction..IO1Direction (bit 10, 6, 2) is set.

In this case the polarity bit IO3Polarity..IO1Polarity (bit 9, 5, 1) in the FCR specifies the output signal level at the corresponding I/O pin.

IOxPolarity = 1 specifies a high level.

IOxPolarity = 0 specifies a low level.

The interrupt mask bit IO3Mask..IO1Mask (bit 8, 4, 0) must be set (disable interrupt), when the corresponding I/O pin is used as output.

*hyper*stone E1-32 Development Board

The *hyper*stone E1-32 Development Board has two on-board LED's driven by pin IO1 and IO2:

IO1 drives the red LED1.

IO2 drives the green LED2.

LED1 and LED2 can be disconnected separately by board jumper group J3 when using the corresponding I/O pin as input.

hyperstone electronics

FCR: Clock Output CLKOUT (only E1-32X)

Unifying RISC and DSP

- Bit 22 of FCR controls polarity of CLKOUT

- Bit 19..18 of FCR controls clock rate of CLKOUT

FCR(19)	FCR(18)	CLKOUT
1	1	static level
1	0	Processor Clock
0	1	Processor Clock : 2
0	0	Processor Clock : 4

hyperstone electronics

Internal Timer

Unifying RISC and DSP

- **Internal Timer**
  - **Timer Prescaler Register TPR**
  - **Timer Register TR**
  - **Timer Compare Register TCR**
  - **Timer Prescaler Register TPR and PLL**

hyperstone electronics

Internal Timer

Unifying RISC and DSP

- **On-chip Timer**
  - controlled via three 32-bit registers:

    Timer Prescaler Register TPR

    Timer Register TR

    Timer Compare Register TCR
  - TR is incremented by one each time unit modulo 2^{32}
  - internal timer interrupt generated when:

    TR ≥ TCR ⇒ result(31..0) := TR(31..0) - TCR(31..0)

    result(31) = 0

    and timer interrupt in FCR enabled
  - internal timer interrupt cleared by:

    loading the TCR with a value > than the current content of the TR
  - A timer delay time in the TCR is calculated according to the formula:

    TCR Value = current content of TR + numbers of delay time units

32

The on-chip timer is controlled via the three registers:

Timer Prescale Register	TPR
Timer Register	TR
Timer compare register	TCR

The TR is a 32-bit register which is incremented by one each time unit modulo 2^{32}. The content of the TCR is compared continuously with the content of the Timer Register TR. When the internal timer interrupt is enabled (bit 23 in FCR cleared) and the value in the TR is higher than or equal to the value in the TCR, a timer interrupt is generated.

This timer interrupt is cleared by loading the TCR with a value higher than the current content of the TR.

The timer interrupt can be masked out by setting bit 23 of the Function Control Register FCR to one (default after Reset). This bit does not affect the timer and compare function.

A timer delay time in the TCR is calculated according to the formula:

TCR Value = current content of TR + number of delay time units

The maximum number of delay time units allowed for this calculation is 2^{31}-1.

hyperstone electronics

Timer Prescaler Register (1)

Unifying RISC and DSP

q **Timer Prescaler Register TPR**

- **Bits 23..16 of the TPR contain the Prescaler Value**
- **Prescaler Value adapts timer clock to different processor clock frequencies:**

$$\text{Frequency of Timer Clock} = \frac{\text{Frequency of Processor Clock}}{\text{Prescaler Value} + 2}$$

- **Prescaler Value is calculated according to the formula:**

  Prescaler Value = (Time Unit * Frequency of Processor Clock) - 2

- **Prescaler Value must be in the range of 0..255**
- **Bits 23..16 are set to zero on Reset**

Timer Prescaler Register (2)

Timer Prescaler Register TPR

- Bits 27..26 of the TPR control internal phased locked loop PLL (only E1-32X)
- PLL provides processor clock rate multiplication of the input clock
- Bits 27..26 are set to 10_2 on Reset

TPR(27)	TPR(26)	Clock Rate Multiplication
1	1	Processor Clock = Clock Input : 2
1	0	Processor Clock = Clock Input (default after Reset)
0	1	Processor Clock = Clock Input x 2
0	0	Processor Clock = Clock Input x 4

Defining Prescaler Value (example)

```
PLLClockDivider EQU   %10 << 26 ; CPU Clock = Clock Input
TimeUnit        EQU    1        ; in microseconds (10^-6)
ProcessorClock  EQU   50        ; in megahertz    (10^6)
PrescalerValue  EQU   ((TimeUnit * ProcessorClock) - 2) << 16
TPRValue        EQU   PLLClockDivider + PrescalerValue
```

hyperstone electronics

Runtime Stack

Unifying RISC and DSP

- **Runtime Stack**
  - **Local Registers and Stack Frame**
  - **Runtime Stack**
  - **Register Stack**

35

hyperstone lectronics

Local Registers and Stack Frame

Unifying RISC and DSP

- **Local Registers**
  - 64 local registers of 32 bits each
  - each local register can be used as operand register, as source register and as destination register of an instruction
  - organized into a 64-word circular register stack to hold subprogram stack frames
- **Stack Frame**
  - a set of up to 16 local registers
  - automatically allocated upon subprogram entry (CALL or TRAP instruction)
  - automatically released upon subprogram return (RET instruction)
  - stack frames can overlap to pass parameters (FRAME instruction)

36

The *hyper***stone** RISC technology is based on a load-store architecture. It is register-oriented and build around a 32-bit wide register stack that holds 64 general purpose local registers. Each local register can be used as operand register, as source register and as destination register of an instruction.

The local registers are organized into a 64-word, circular register stack to hold subprogram stack frames. A stack frame is a set of up to 16 local registers, its registers can be addressed by an instruction as L0..L15.

The Call instruction and the Trap instruction causes a branch to a subprogram. These instructions create a new stack frame with a length of six local registers. The contents of the global program counter register PC and the global status register SR are automatically saved into the first two registers (L0 and L1) of the new stack frame.

The Return instruction returns control from a subprogram entered through a Call or a Trap to the instruction located at the return address and restores the status from the saved return status. The Return instruction releases the current stack frame and restores the preceding stack frame.

A Frame instruction restructures the current stack frame. The current stack frame can overlap with the previous stack frame at a variable range to pass parameters between two subprograms.

hyperstone electronics

Runtime Stack (1)

Unifying RISC and DSP

- **Runtime Stack**
  - divided into a memory part (hardware stack) and a register part (register stack)
- **Register Part of Runtime Stack (register stack)**
  - holds most recent stack frames
  - current stack frame is always kept in the register part of the stack
  - frame pointer FP points to the first register of the current stack frame (addressed as local register L0)
  - frame length FL indicates the number of registers (maximum 16) assigned to the current stack frame

37

The runtime stack holds generations of stack frames in last-in-first-out order and is divided into a memory part and a register part.

The register part of the stack, implemented by the 64 local registers organized as a circular buffer, holds the most recent stack frames. The current stack frame is always kept in the register part of the stack.

The frame pointer FP points to the first register of the current stack frame (addressed as register L0). All registers of a stack frame are addressed relative to this pointer. The frame length FL indicates the number of local registers (maximum 16) assigned to the current stack frame. FP and FL are part of the global status register SR.

The real-time operating system hyRTK supports multiple runtime stacks. Each user task (stack-level task) has its own runtime stack.

hyperstone electronics

Runtime Stack (2)

Unifying RISC and DSP

q **Memory Part of Runtime Stack (hardware stack)**

- stack frames are pushed to the memory part of the runtime stack, if the register stack overflows
- stack frames are popped from the memory part of the runtime stack, if the register stack underflows
- overflow and underflow of the register stack is managed automatically
- global stack pointer register SP contains the address of the first free memory location + 4 in which the first local register would be saved by a push operation
- global upper stack bound register UB guards the memory part of the runtime stack

38

Stack frames are automatically pushed to the memory part of the runtime stack, if the register stack overflows. Stack frames are automatically popped from the memory part of the runtime stack, if the register stack underflows.

The global stack pointer register SP contains the top address + 4 of the memory part of the stack, that is the address of the first free memory location in which the first local register would be saved by a push operation to the memory part of the runtime stack.

The memory part of the runtime stack grows from low to high address and is guarded by the global upper stack bound register UB. The UB contains the address beyond the highest legal memory stack location. It is used by the Frame instruction to inhibit stack overflow.

A small stack space can be reserved above UB. UB can then be set to a higher value by a Frame Error handler to free stack space for error handling.

hyperstone electronics

Register Stack (1)

Unifying RISC and DSP

```
A:  FRAME   L9, L0      ; set frame length FL = 9
          :
    code of function A
          :             ; L7 and L8 contain parameters to pass B
    CALL    L9, 0, B    ; call function B
          :
    code of function A
          :
    RET     PC, L0      ; return to function calling A, restore frame

B:  FRAME   L11, L2     ; set frame length FL = 11, decrement FP by 2
          :             ; passed parameter1 can now be addressed in L0
          :             ; passed parameter2 can now be addressed in L1
          :
    code of function B
          :
    RET     PC, L2      ; return to function A, frame A is restored by
                        ; copying return PC and return SR in L2 and L3
                        ; of frame B to PC and SR
```

39

Because the complete stack management is accomplished automatically by the hardware, programming the stack handling instructions is easy and does not require any knowledge of the internal working of the stack.

The above example demonstrate how the Call, Frame and Return instructions are applied to achieve the stack behaviour of the register part of the stack shown in the next figure.

A currently activated function A has a frame length of FL = 9. A call to function B needs 2 parameters to be passed. The parameters are placed by function A in registers L7 and L8 before calling B. The Call instruction addresses L9 as destination for the return PC and return SR register pair to be used by function B on return to function A.

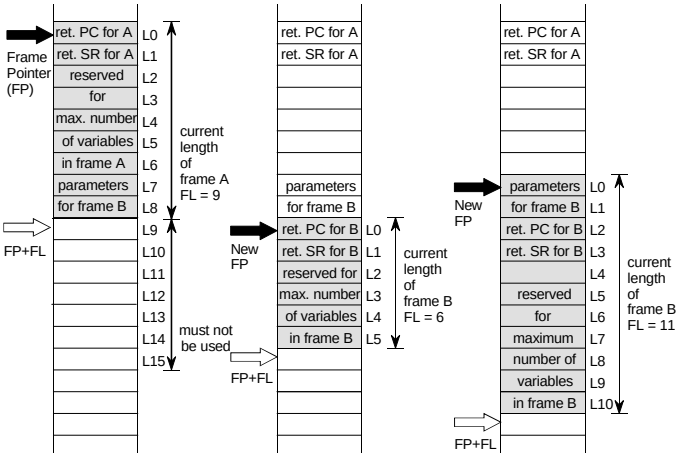
On entry of function B, the new frame of B has an implicit length of FL = 6. It starts physically at the former register L9 of frame A. However, since the frame pointer FP has been incremented by 9 by the Call instruction, this register location is now being addressed as L0 of frame B. The passed parameters cannot be addressed because they are located below the new register L0 of frame B. To make them addressable, a Frame instruction decrements the frame pointer FP by 2. The frame instruction must be executed immediately after the preceding Call instruction, otherwise an Interrupt, Parity Error, Extended Overflow or Trace exception could seperate the Call from the corresponding Frame instruction before the frame pointer FP is decremented to include the passed parameters.

hyperstone electronics

Register Stack (2)

Unifying RISC and DSP

```
A:  FRAME   L9, L0
    ...
    CALL    L9, 0, B
    ...
    RET     PC, L0

B:  FRAME   L11, L2
    ...
    RET     PC, L2
```



after FRAME L9, L0 **after CALL L9, 0, B** **after FRAME L11, L2**

hyperstone electronics

Miscellaneous Topics

Unifying RISC and DSP

- Privilege States
- Trap Entry Table
- Interrupt-Lock Flag L
- Global Registers
  - Global Registers
  - High Global Flag
- Supervisor State
- Runtime Stack Initialization
- Power-Down Mode
- Sleep Mode

hyperstone electronics

Privilege States

Unifying RISC and DSP

- Supervisor State Flag S of the Status Register SR controls Privilege State
  - User State S = 0
  - Supervisor State S = 1
- Entering into Supervisor State
  - Executing a Trap

    Trap 0 .. Trap 63
  - Exception Processing (ordered by priority)

    Reset

    Range, Pointer, Frame and Privilege Error

    Extended Overflow

    Parity Error

    Interrupt and Timer Interrupt

    Trace Exception

hyperstone electronics

Trap Entry Table (1)

Unifying RISC and DSP

- **Trap Entry Table contains up to 64 entries**
- **Entries of the Trap Entry Table are intended to each contain an instruction branching to the associated function.**
- **Spacing of the entries is 4 bytes**
- **Trap Entries TRAP 0 .. TRAP 55**

Address	Trap Entry	Description	Example of Instruction
FFFF FF00	TRAP 0		`TRAP0: MOVI PC, #Trap0`
FFFF FF04	TRAP 1		`TRAP1: MOVI PC, #Trap1`
:	:		:
FFFF FFC0	TRAP 48	IO2 Interrupt -- priority 15	`TRAP48: MOVI PC, #IO2Interrupt`
FFFF FFC4	TRAP 49	IO1 Interrupt -- priority 14	`TRAP49: MOVI PC, #IO1Interrupt`
FFFF FFC8	TRAP 50	INT4 Interrupt -- priority 13	`TRAP50: MOVI PC, #INT4Interrupt`
FFFF FFCC	TRAP 51	INT3 Interrupt -- priority 11	`TRAP51: MOVI PC, #INT3Interrupt`
FFFF FFD0	TRAP 52	INT2 Interrupt -- priority 9	`TRAP52: MOVI PC, #INT2Interrupt`
FFFF FFD4	TRAP 53	INT1 Interrupt -- priority 7	`TRAP53: MOVI PC, #INT1Interrupt`
FFFF FFD8	TRAP 54	IO3 Interrupt -- priority 5	`TRAP54: MOVI PC, #IO3Interrupt`
FFFF FFDC	TRAP 55	Timer Interrupt -- priority selectable as 6, 8, 10, 12	`TRAP55: MOVI PC, #TimerInterrupt`

hyperstone electronics

Trap Entry Table (2)

Unifying RISC and DSP

- **Trap Entries TRAP 56 .. TRAP 63**

Address	Trap Entry	Description		Example of Instruction
FFFF FFE0	TRAP 56	Reserved	-- priority 17 (lowest)	`TRAP56: MOVI PC, #TrapReservecd`
FFFF FFE4	TRAP 57	Trace Exception	-- priority 16	`TRAP57: MOVI PC, #TraceException`
FFFF FFE8	TRAP 58	Parity Error	-- priority 4	`TRAP58: MOVI PC, #ParityError`
FFFF FFEC	TRAP 59	Extended Overflow	-- priority 3	`TRAP59: MOVI PC, #OverflowError`
FFFF FFF0	TRAP 60	Range, Pointer, Frame and Privilege Error	-- priority 2	`TRAP60: MOVI PC, #MiscError`
FFFF FFF4	TRAP 61	Reserved	-- priority 1	`TRAP61: MOVI PC, #TrapReservecd`
FFFF FFF8	**TRAP 62**	**Reset**	**-- priority 0 (highest)**	`TRAP62: MOVI PC, #ResetEntry`
FFFF FFFC	TRAP 63	Error entry for instruction code of all ones		`TRAP63: MOVI PC, #AllOnesError`

- **Bits 14..12 of the MCR map the Trap Entry Table to one of the memory areas MEM0..MEM3 or the IRAM**
- **Trap Entry Table is mapped to the end of memory area MEM3 after Reset**

hyperstone electronics

Interrupt-Lock Flag L

Unifying RISC and DSP

- Interrupt-Lock Flag L of Status Register SR controls Exception Inhibition

  L = 1 inhibits exceptions

  Interrupt
  Parity Error
  Extended Overflow

- Interrupt-Lock Flag L is automatically set to one by any Exception
- Interrupt-Lock Flag L can not be set to one in User State

hyperstone electronics

Global Registers

Unifying RISC and DSP

- **16 Global Registers**

G0	Program Counter PC
G1	Status Register SR
G2	Floating-point Exception Register FER
G3..G15	General purpose registers

- **16 High Global Registers**

G16..G17	Reserved
G18	Stack Pointer SP
G19	Upper Stack Bound UB
G20	Bus Control Register BCR
G21	Timer Prescaler Register TPR
G22	Timer Compare Register TCR
G23	Timer Register TR
G24	Watchdog Compare Register WCR
G25	Input Status Register ISR
G26	Function Control Register FCR
G27	Memory Control Register MCR
G28..G31	Reserved

hyperstone electronics

Global Registers and High Global Flag

Unifying RISC and DSP

- High Global Flag H of Status Register SR controls access to Global Registers

- High Global Flag is effective only the first cycle of the next instruction after it was set
- High Global Flag is cleared automatically
- Only the MOV or MOVI instruction can be used to access a High Global Register

```
ORI      SR, 1<<5     ; set high global flag H in Status Register
MOVI     G7, 0        ; access high global register G23 (Timer Register)
MOVI     G7, 0        ; access global register G7

ORI      SR, 1<<5     ; set high global flag H in Status Register
MOVI     TR, 0        ; access high global register G23 (Timer Register)

ORI      G1, 1<<5     ; set high global flag H in Status Register
MOV      L0, G23      ; access high global register G23 (Timer Register)
```

hyperstone electronics

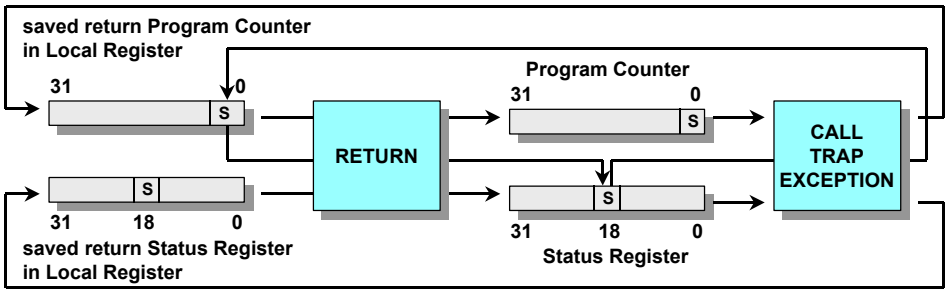
Supervisor State

Unifying RISC and DSP

q **Privileged to be executed only in Supervisor State**

Supervisor State Flag S = 1

- **Copying an operand to any of the high global registers**
- **Changing the interrupt-lock flag L from zero to one**
- **Returning through a Return instruction to Supervisor State**



48

The Supervisor State Flag S does not affect the behaviour of the Program Counter PC, since program instructions are located on halfword boundaries. Bit zero of the PC is always interpreted as zero by the instruction execution unit.

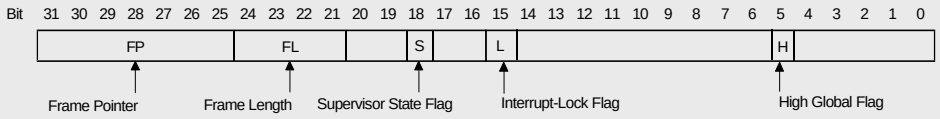
hyperstone electronics

Runtime Stack Initialization (1)

Unifying RISC and DSP

- **Stack pointer register SP, upper stack bound register UB and frame pointer FP must be initialized, before a Call or Trap instruction can be executed, since the register stack is in an undefinded state after Reset**
- **Bits 31..25 of the status register SR represent the frame pointer FP.**
- **Least significant six bits of the FP (bits 30..25) must point to the beginning of the current stack frame in the register stack, that is, they point to L0**
- **Frame pointer FP must contain bits 8..2 of the stack pointer register SP**
- **Bits 24..21 of the status register SR represent the frame length FL of the current stack frame**
- **Frame length FL = 0 is always interpreted as FL = 16**

Status Register SR (global register G1)



hyperstone electronics

Runtime Stack Initialization (2)

Unifying RISC and DSP

- **The following code sequence shows an example which can be used to set up SP, UB and FP after Reset**

```
StackBase   EQU     $C000000                ; base address of hardware stack
StackSize   EQU     $1000                   ; stack size of hardware stack

AfterReset:
            ORI     SR, 1<<5                ; set high global flag H
            MOVI    SP, StackBase           ; set base address of hardware stack

            ORI     SR, 1<<5                ; set high global flag H
            MOVI    UB, StackBase+StackSize ; set upper bound of stack

            MOVI    L0, StackInitialized    ; initialize return PC
            ORI     L0, 1                   ; set supervisor state flag S in L0
                                            ; -> return to supervisor state

            MOVI    L1, StackBase<<(25-2)   ; bits 31..25 of SR contain
                                            ; bits 8..2 of SP
            ORI     L1, 1<<15               ; set interrupt-lock flag L in L1
                                            ; -> disable all interrupts
            RET     PC, L0                  ; restore saved PC and saved SR
                                            ; located in L0 and L1
StackInitialized:
            FRAME  L0, L0                   ; runtime stack is set up
            ...
```

50

The memory part of the runtime stack is located in this example in the internal RAM of the *hyper***stone** E1-32 (memory address C000 0000_{16}).

The frame pointer FP can only be set by returning to supervisor state through a return instruction. The supervisor state flag S is saved in bit zero of the saved return PC of the current stack frame (L0 in the above example). The Return instruction restores the saved S flag from this bit position to the S flag in bit position 18 of the SR (thereby overwriting the bit 18 returned from the saved return SR).

After Reset, the interrupt-lock flag L (bit 15 of the status register SR) is set. When the L flag is one, all Interrupt, Parity Error and Extended Overflow exceptions are inhibited. Changing the L flag from zero to one is privileged to supervior or return from supervisor to supervisor state. A trap to Privilege Error occurs if the L flag is set under program control from zero to one in user state. The L flag is set to one by any exception (e.g. Reset, Interrupt, etc.).

The L flag is set in the return SR, since all interrupts should be locked out after initialization of the runtime stack. The Return instruction restores the saved status register (L1 in the above example) to the SR.

hyperstone electronics

Power-Down Mode

Unifying RISC and DSP

q **Power-Down Mode**

- **Power-Down Mode is entered by a 1-to-0 transition of MCR(22)**
- **Execution Pipeline is halted**
- **Clocked Logic**
  - **Timer**
  - **IO3 control modes**
  - **Interrupt**
  - **DRAM refresh**
  - **IRAM refresh**
- **Resumes execution at**
  - **any Interrupt**
  - **Reset (external, watchdog)**

Supply Voltage:	5V	
Clock Frequency [MHz]	Power consumption typical [mW]	Power consumption power-down [mW]
10	200	5
25	380	12
33	470	16
40	540	20
50	650	25
66	800	33

hyperstone electronics

Sleep Mode (only E1-32X)

Unifying RISC and DSP

q **Sleep Mode**

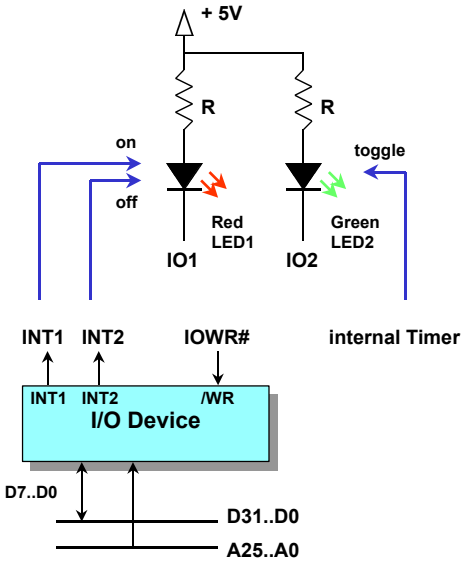
- **Sleep Mode is entered via**
  I/O Write: A(27) = 1, A(25..22) = 1
- **Processor Clock is switched off**
- **Content will be lost during sleep mode**
  Timer count
  internal RAM
  DRAM
- **Resumes execution at**
  any Interrupt
  External Reset
- **After Processor awakes it continues with standard reset procedure**

hyperstone electronics

Assembler Example (without Real-Time Operating System)

Unifying RISC and DSP

- q **2 LED s**
  - IO1 drives red LED1
  - IO2 drives green LED2
- q **I/O Device**
  - INT1 switches red LED1 on
  - INT2 switches red LED1 off
  - Interrupt INT1 and INT2 can be cleared via write access
- q **Internal Timer**
  - toggles green LED2 each 1 second
- q **Initialization of E1-32**
  - BCR, MCR and FCR
  - Run-time stack
  - internal Timer
- q **Used Software Development Tools**
  - Macro Assembler hyMasm
  - Linker hyLink
  - EPROM Formatter hyEPROM



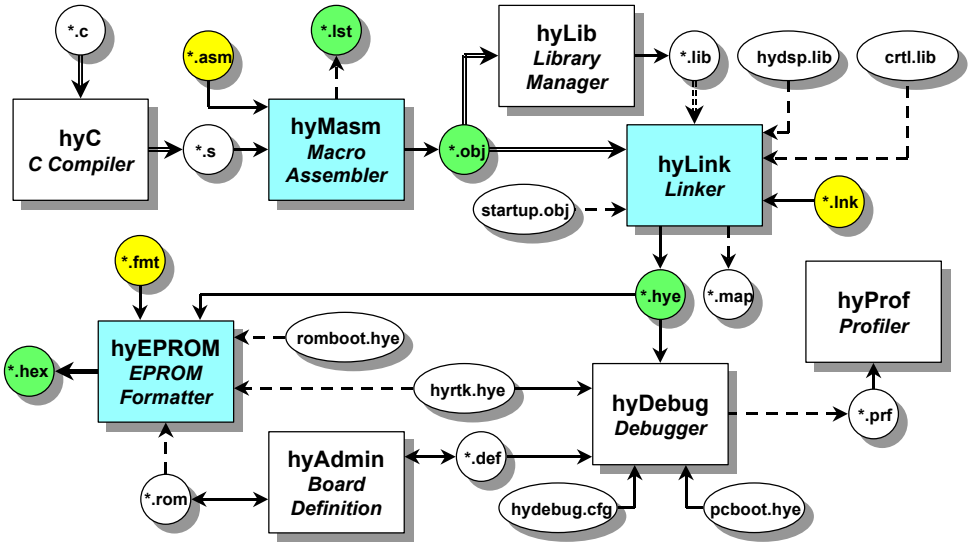
53

hyperstone electronics

Software Tools

hyMasm, hyLink, hyEPROM

Unifying RISC and DSP



54

Assembler Segment: EntryTable

Unifying RISC and DSP

q **entrytab.asm**

```
XREF       InterruptINT1   ; turns red LED1 on (connected to IO1)
XREF       InterruptINT2   ; turns red LED1 off
XREF       TimerInterrupt  ; toggles green LED2 (connected to IO2)
XREF       ResetEntry
XREF       TrapNotUsed
XREF       SetPowerDown

SEGMENT TRAP17
TRAP17:    MOVI   PC, #SetPowerDown        ; located at address FFFF FF44

SEGMENT EntryTable
TRAP48:    MOVI   PC, #TrapNotUsed         ; located at address FFFF FFC0
TRAP49:    MOVI   PC, #TrapNotUsed         ; located at address FFFF FFC4
TRAP50:    MOVI   PC, #TrapNotUsed         ; located at address FFFF FFC8
TRAP51:    MOVI   PC, #TrapNotUsed         ; located at address FFFF FFCC
TRAP52:    MOVI   PC, #InterruptINT2       ; located at address FFFF FFD0
TRAP53:    MOVI   PC, #InterruptINT1       ; located at address FFFF FFD4
TRAP54:    MOVI   PC, #TrapNotUsed         ; located at address FFFF FFD8
TRAP55:    MOVI   PC, #TimerInterrupt      ; located at address FFFF FFDC
TRAP56:    MOVI   PC, #TrapNotUsed         ; located at address FFFF FFE0
TRAP57:    MOVI   PC, #TrapNotUsed         ; located at address FFFF FFE4
TRAP58:    MOVI   PC, #TrapNotUsed         ; located at address FFFF FFE8
TRAP59:    MOVI   PC, #TrapNotUsed         ; located at address FFFF FFEC
TRAP60:    MOVI   PC, #TrapNotUsed         ; located at address FFFF FFF0
TRAP61:    MOVI   PC, #TrapNotUsed         ; located at address FFFF FFF4
TRAP62:    MOVI   PC, #ResetEntry          ; located at address FFFF FFF8
TRAP63:    MOVI   PC, #TrapNotUsed         ; located at address FFFF FFFC
END TRAP62
```

55

The **XREF** directive specifies that the label in the operand field is a label defined in another module. The reference will be resolved by the linker. The label must not be defined in the current module.

Syntax:

`XrefDirective ::= XREF Label`

A **SEGMENT** assembler directive defines where the following code or data is to be placed at link time. A **SEGMENT** directive consists of the reserved word **SEGMENT** followed by an identifier denoting the *segment name*.

When assembling a source file, the assembler places all code and/or data in the current segment until the next segment directive is encountered.

Up to 64 different segments may be used in a program. This allows for great flexibility and meets all requirements even for large systems.

Syntax:

`SegmentDirective ::= SEGMENT Identifier`

The **END** directive informs the cross-assembler of the end of the source input file. If the optional Label behind **END** is present, it is used as the start address of the program. This start address is included in the object file and passed to the linker. During the linking process, only one module may have a start address, otherwise an error results.

Any text found after an **END** directive is ignored.

Syntax:

`EndDirective ::=[LabelDefinition] END [Label]`

Assembler Segment: ResetSegment (1)

Unifying RISC and DSP

- **reset.asm (part 1)**

```
INCLUDE "system.inc"

XDEF ResetEntry

XREF InitializationReady    ; after initialization branch to this address
XREF FCRVariable            ; variable to store write-only FCR


SEGMENT ResetSegment
ResetEntry:

; all interrupts are inhibited,
; because interrupt-lock flag L is set after Reset
; instruction execution in Supervisor State after Reset
...
```

56

The **INCLUDE** assembler directive allows the insertion of source code from another file into the current source file during assembly. The included file is assembled into the current source file immediately after the directive. When the **EOF** (end-of-file) of the included file is reached, the assembly resumes on the line after the include directive.

The file to be included is named in the string constant after the **INCLUDE** directive. A file name may contain a path. If the file does not exist, an error results and the assembly is aborted. Recursive includes also result in an error.

The assembler hyMasm searches for source files to be included in the current directory and in each directory listed in the MS-DOS environment variable **HYGCCINC**.

Syntax:

`IncludeDirective ::= INCLUDE StringConstant`

The **XDEF** directive defines a label in the current module as an external symbol that is to be made visible to other modules at link time. The operand must reference a label which is defined anywhere in the assembly file.

Syntax:

`XdefDirective ::= XDEF Label`

Assembler Segment: VariablesSegment

Unifying RISC and DSP

- **var.asm**

```
XDEF       FCRVariable

SEGMENT VariablesSegment
FCRVariable: D.WU         ; variable to store write-only 32 bit FCR


END
```

57

Data declaration directives are used to allocate memory. Two types of data storage are allowed, scalar and array. The following table shows the available data declaration types, the corresponding data types and the alignment rules:

Type	Data Type	Alignment
D.BU	unsigned byte	byte boundary
D.BS	signed byte	byte boundary
D.BC	character string	byte boundary
D.HU	unsigned halfword	halfword boundary
D.HS	signed halfword	halfword boundary
D.WU	unsigned word	word boundary
D.WS	signed word	word boundary
D.WF	single- precision floating-point	word boundary
D.DF	double-precision floating-point	word boundary

The assembler automatically aligns data based on its data type. All labels denoting data declaration directives are automatically adjusted to denote the exact begin of the corresponding data declaration.

Syntax:

```
ScalarDeclaration ::= Type [ConstExpression]
Type ::= D.BU | D.BS | D.BC |
         D.HU | D.HS |
         D.WU |D.WS  | D.WF |
         D.DF
```

A single data element of the specified type is reserved. The memory location may be initialized.

hyperstone electronics

System Include File

Unifying RISC and DSP

- **system.inc**

```
BCRValue           EQU $F37505CB      ; specify according to the connected hardware
MCRValue           EQU $FDD9F0F0      ; specify according to the connected hardware
PowerDown          EQU 1<<22          ; power-down bit in MCR

PLLClockDivider EQU %10 << 26   ; CPU Clock = Clock Input
TimeUnit           EQU  1             ; in microseconds (10^-6)
ProcessorClock     EQU 50             ; in megahertz    (10^6)
PrescalerValue     EQU ((TimeUnit * ProcessorClock) - 2) << 16
TPRValue           EQU PLLClockDivider + PrescalerValue
TimerInterval      EQU 1000000        ; 1 000 000 microseconds

StackBase          EQU $C0000000      ; first address of IRAM
StackSize          EQU $400           ; size: 1Kbyte

FCRValue          EQU $CF7FFF99
IO1Polarity       EQU (%1 << 1)
IO2Polarity       EQU (%1 << 5)

PeripheralAddr    EQU $03FFF7F8      ; specify according to the connected hardware
ClearINT1         EQU $F             ; value to clear INT1
ClearINT2         EQU $0             ; value to clear INT2
```

Assembler Segment: ResetSegment (2)

- reset.asm (part 2)

```
; initialize BCR and MCR
; enable refresh of the IRAM
ORI    SR, 1<<5                ; set high global flag H
MOVI   BCR, BCRValue           ; set BCR

ORI    SR, 1<<5                ; set high global flag H
MOVI   MCR, MCRValue           ; set MCR

; initialize TPR and TR
ORI    SR, 1<<5                ; set high global flag H
MOVI   TPR, TPRValue           ; set timer prescaler register
ORI    SR, 1<<5                ; set high global flag H
MOVI   TR, 0                   ; set timer register

...
```

Assembler Segment: ResetSegment (3)

Unifying RISC and DSP

- reset.asm (part 3)

```
; enable external interrupt INT1 and INT2
; set polarity of INT1 and INT2 to non-inverted
; enable internal timer interrupt
; set priority of timer interrupt to 6
; set IO1 and IO2 to output state
; set polarity of IO1 and IO2 to Inverted
; interrupts are still inhibited

MOVI   L0, FCRValue
STW.A  0, L0, FCRVariable      ; store FCRValue in FCRVariable

ORI    SR, 1<<5                ; set high global flag H
MOV    FCR, L0                 ; set FCR
...
```

60

The content of the Function Control Register FCR is saved in the variable `FCRVariable`, since this register is write-only.

Absolute Address Mode:

Notation load instruction: **`LD`**`xx.`**`A`** `0, Rs, dis`

Notation store instruction: **`ST`**`xx.`**`A`** `0, Rs, dis`

Data Type `xx` is with:

`BU`: byte unsigned; **`HU`**: halfword unsigned; **`W`**: word;

`BS`: byte signed; **`HS`**: halfword signed; **`D`**: double-word;

The displacement `dis` is used as an address into memory address space.

In the case of all data types except byte, address bit zero of `dis` is treated as zero.

The displacement `dis` provides absolute addressing at the beginning and the end of the memory.

Assembler Segment: ResetSegment (4)

Unifying RISC and DSP

- **reset.asm (part 4)**

```
; initialize run-time stack
ORI    SR, 1<<5                ; set high global flag H
MOVI   SP, StackBase           ; set base address of stack

ORI    SR, 1<<5                ; set high global flag H
MOVI   UB, StackBase+StackSize ; set upper bound of stack

MOVI   L0, InitializationReady ; initialize return PC
MOVI   L1, StackBase<<(25-2)   ; bits 31..25 of SR contain bits 8..2 of SP

RET    PC, L0                  ; restore saved PC and SR located in L0 and L1
                               ; supervisor state flag S is not set in L0
                               ; interrupt-lock flag L is not set in L1
END
```

61

After the hardware is initialized, instruction execution continues at `InitializationReady`.

hyperstone electronics

Assembler Segment: MainSegment

Unifying RISC and DSP

- **main.asm**

```
XDEF    InitializationReady

SEGMENT MainSegment

InitializationReady:
FRAME  L0, L0                  ; 16 Registers in stack frame
                               ; interrupt-lock flag is now cleared
                               ; runtime stack is set up
WaitForInterrupt:
TRAP   17                      ; set power-down mode
BR     WaitForInterrupt        ; looping forever

END
```

Assembler Segment: InterruptSegment (1)

Unifying RISC and DSP

- intr.asm (part 1)

```
INCLUDE "system.inc"

XDEF       InterruptINT1  ; turns red LED1 on (connected to IO1)
XDEF       InterruptINT2  ; turns red LED1 off
XDEF       TimerInterrupt ; toggles green LED2 (connected to IO2)
XDEF       TrapNotUsed
XDEF       SetPowerDown
XREF       FCRVariable

SEGMENT InterruptSegment

SetPowerDown:
FRAME   L3, L0
MOVI    L2, MCRValue
ORI     SR, 1<<5           ; set high global flag H
MOV     MCR, L2            ; set power-down bit from 0 to 1
ANDNI   L2, 1<<22          ; set power-down mode
ORI     SR, 1<<5           ; set high global flag H
MOV     MCR, L2            ; power down is set, program stops
RET     PC, L0             ; return is executed after power up

TrapNotUsed:               ; referenced in Entry Table
FRAME L2, L0               ; FL = 2; L0 = return PC, L1 = return SR
RET   PC, L0
```

Assembler Segment: InterruptSegment (2)

Unifying RISC and DSP

- intr.asm (part 2)

```
TimerInterrupt:
FRAME L3, L0                        ; FL = 3; L0 = return PC, L1 = return SR

LDW.A   0, L2, FCRVariable          ; load FCRVariable
XORI    L2, IO2Polarity             ; toggle IO2Polarity bit
STW.A   0, L2, FCRVariable          ; store FCRVariable

ORI     SR, 1<<5                    ; set high global flag H
MOV     FCR, L2                     ; set FCR

ORI     SR, 1<<5                    ; set high global flag H
MOV     L2, TR                      ; move content of TR to local register L0
ADDI    L2, TimeUnit*TimerInterval  ; add timer delay time to local register L0
ORI     SR, 1<<5                    ; set high global flag H
MOV     TCR, L2                     ; set new TCR value

RET     PC, L0
```

hyperstone electronics

Assembler Segment: InterruptSegment (3)

Unifying RISC and DSP

- intr.asm (part 3)

```
InterruptINT1:                ; turns red LED1 on (connected to IO1)
FRAME   L3, L0                ; FL = 3; L0 = return PC, L1 = return SR
MOVI    L2, ClearINT1         ; value to clear INT1
STW.IOA 0, L2, PeripheralAddr ; clear INT1 with I/O write access

LDW.A   0, L2, FCRVariable    ; load FCRVariable
ORI     L2, IO1Polarity       ; set IO1Polarity bit
STW.A   0, L2, FCRVariable    ; store FCRVariable

ORI     SR, 1<<5              ; set high global flag H
MOV     FCR, L2               ; set FCR

RET     PC, L0
```

Assembler Segment: InterruptSegment (4)

Unifying RISC and DSP

- **intr.asm (part 4)**

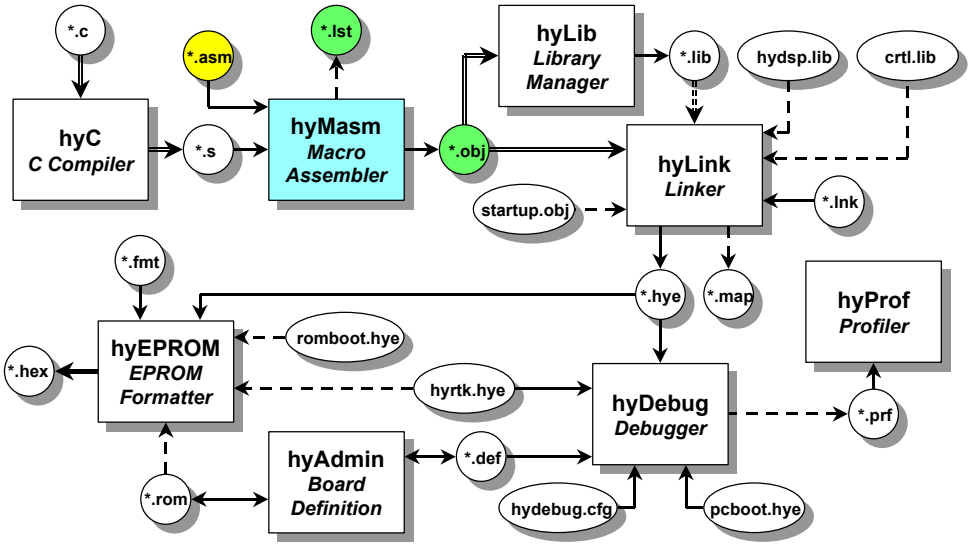
```
InterruptINT2:                ; turns red LED1 off (connected to IO1)
FRAME   L3, L0                ; FL = 3; L0 = return PC, L1 = return SR
MOVI    L2, ClearINT2         ; value to clear INT2
STW.IOA 0, L2, PeripheralAddr ; clear INT2 with I/O write access

LDW.A   0, L2, FCRVariable    ; load FCRVariable
ANDNI   L2, IO1Polarity       ; clear IO1Polarity bit
STW.A   0, L2, FCRVariable    ; store FCRVariable

ORI     SR, 1<<5              ; set high global flag H
MOV     FCR, L2               ; set FCR

RET     PC, L0

END
```

hyperstone electronics

Software Tools

hyMasm

Unifying RISC and DSP



67

hyperstone electronics

Assembler hyMasm (1)

Unifying RISC and DSP

- Translates *hyper*stone assembly language source programs into relocatable object modules
- Command line invocation

  `hymasm {options} filename`
- Options

  `-LIST`

  generates output listing file with `.lst` filename extension

  `-DSymbol[=IntVal]`

  defines the symbol `Symbol` together with optional value `IntVal`

  the symbol is treated the same as an `EQU` directive in the source code

  `-g`

  generates debug information for debugger hyDebug (only with hyRTK)

  `-QUIET`

  displaying screen title and copyright information is suppressed

68

The five assembly language source files of the preceding example can be translated into object modules as follows:

```
hymasm -LIST entrytab
hymasm -LIST reset
hymasm -LIST main
hymasm -LIST intr
hymasm -LIST var
```

The above command line can be entered with any combination of lower-case or upper-case characters. Options may be specified in any order but must precede `filename`.

If an extension is not specified on `filename`, then **`.asm`** is assumed. The extension **`.obj`** and **`.lst`** can not be used for source input files to prevent accidental overwriting of assembler source and listing files by the assembler itself.

An object file, `filename`**`.obj`**, is created automatically when no errors in the source program are detected. The old object file, if any, is always renamed to `filename`**`.obb`** regardless of wether errors have been detected or not.

A comprehensive output listing file, `filename`**`.lst`**, containing the source and object code generated, is created when the assembler is invoked with the **`-LIST`** option.

hyperstone electronics

Assembler hyMasm (2)

Unifying RISC and DSP

- **Assembler output listing file (*.lst)**

```
Hyperstone Macro Assembler   Version 4.18     97-08-28     11:06:24    page:   1

PC        Machine Code        I  Line    File: example.asm
00000000                      A     1    EQUValue   EQU  $F
00000000                      A     2
00000000                      A     3    XREF XREFVariable
00000000                      A     4    XDEF XDEFVariable
00000000                      A     5
00000000 FFFF FFFF            A     6    XDEFVariable: D.WU $FFFFFFFF
00000004                      A     7
00000004                      A     8    Convert MACRO RegNo
00000004                      A     9            ORI   \RegNo, EQUValue
00000004                      A    10            ENDMACRO Convert
00000004                      A    11
00000004 6701 0000 0000       A    12    MOVI   L0, XDEFVariable
0000000A 6701 0000 0000       A    13    MOVI   L0, XREFVariable
00000010 9910 B000 0000       A    14    STW.A  0, L0, XDEFVariable
00000016 9910 B000 0000       A    15    STW.A  0, L0, XREFVariable
0000001C 7A0F                 A    16    ORI    L0, $F
0000001E 7A0F                 A    17    ORI    L0, EQUValue
00000020                      A    18    Convert L0
00000020 7A0F                 A+   18            ORI   L0, EQUValue
00000022                      A+   18            ENDMACRO Convert
```

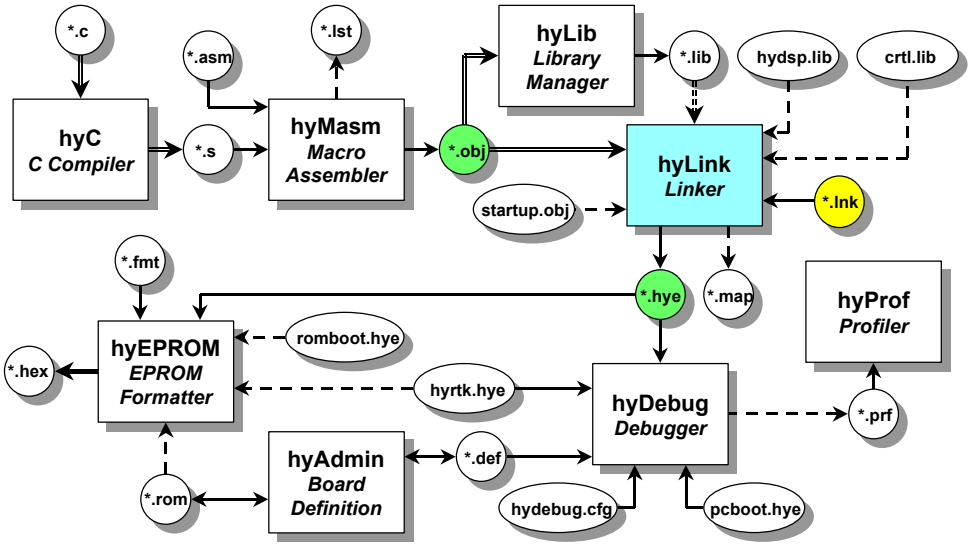
Location Counter | Machine Code | Include Level Indicator (A..Z) | Line Number | Soucre Code

hyperstone electronics

Software Tools

hyLink

Unifying RISC and DSP



70

hyperstone electronics

Linker hyLink

Unifying RISC and DSP

- Linker and locator for object modules created by the hyMasm assembler
- Command line invocation

  `hylink {options} @commandfilename[.lnk]`
- Options

  `-MAP`

  creates a map file `commandfilename.map`, containing information about the location of individual segments and a list of identifiers declared as externals (`XDEF`)

  `-NODEBUG`

  no debug information is included in the *hyper*stone executable file even if some or all of the object files are assembled with debug information

  `-QUIET`

  displaying screen title and copyright information is suppressed

71

The five object modules former created by the hyMasm assembler can be linked as follows:

```
hylink -MAP example1.lnk
```

The above command line can be entered with any combination of lower-case or upper-case characters. Options may be specified in any order but must precede `commandfilename`. If an extension is not specified on `commandfilename`, then `.lnk` is assumed.

Modules are linked in the order specified by the user. Modules to be linked are object files created by the hyMasm assembler. An *hyper*stone executable file is created.

hyperstone electronics

Linker Command File (1)

Unifying RISC and DSP

q **File: example1.lnk (part 1)**

```
; linker command file for assembler example

; the executable file example1.hye is created
; the content of the listed object files are linked into the
; executable file example.hye
; LINK Command
example1.hye = entrytab.obj, reset.obj, main.obj, intr.obj, var.obj

; the XDEF symbols StackBase and StackSize are defined
; DEFINE Command
define StackBase = $C0000000 ; base address of runtime stack
define StackSize = $400      ; stack size of runtime stack
```

72

The above link command file `example1.lnk` contains the following commands:

LINK Command

Syntax:

```
exefilename = objectfilename     | libfilename
              {[,]objectfilename | libfilename}
```

An executable file `exefilename` is created. `objectfilename` is the name of an object file, `libfilename` is the name of a library file. When the name extension of `exefilename` is omitted, **`.hye`** is used by default. The contents of the listed object or library files are linked into the executable file. When the name extension of the object or library file is omitted, **`.obj`** is used by default.

DEFINE Command

Syntax:

```
DEFINE identifier = expression
```

The **`DEFINE`** command creates a user defined **`XDEF`** symbol. The **`XDEF`** identifier and expression must be specified.

identifier is the identifier of the **`XDEF`** created.

expression is the value assigned to the **`XDEF`** created.

hyperstone electronics

Linker Command File (2)

Unifying RISC and DSP

q **File: example1.lnk (part 2)**

```
; the segments ResetSegment, MainSegment, InterruptSegment and
; EntryTable are grouped together to the new segment MEM3Segment
; GROUP Command
GROUP MEM3Segment = ResetSegment, MainSegment, InterruptSegment

; segment EntryTable begins at address ($100000000 - LENGTH OF EntryTable)
; last address of EntryTable is located at address FFFFFFFC16
LOCATE EntryTable AT ($100000000 - LENGTH OF EntryTable)
LOCATE  TRAP17       AT  $FFFFFF44
LOCATE MEM3Segment AT ($FFFFFF44 - LENGTH OF MEM3Segment)
; the segment VariablesSegment begins at address StackBase+StackSize
LOCATE VariablesSegment AT (StackBase+StackSize)
; end of linker command file example1.lnk
```

73

GROUP Command

Syntax:

GROUP segmentname **=** segmentname {[,]segmentname}

The segments segmentname behind the equal (=) character are grouped together to a single *segment group*; this group can then be referenced as one segment by the segment name segmentname preceding the equal (=) character. The segments names making up the new group are then no longer visible to the linker and cannot be used in further linker commands.

The **GROUP** command forces the linker to group the segments in the order specified.

segmentname specifies a segment.

LOCATE Command

Syntax:

LOCATE segmentname **AT** expression

The **LOCATE** command specifies the address at which a segment begins. If multiple locate commands specify overlapping segments, a warning is issued.

segmentname specifies the segment.

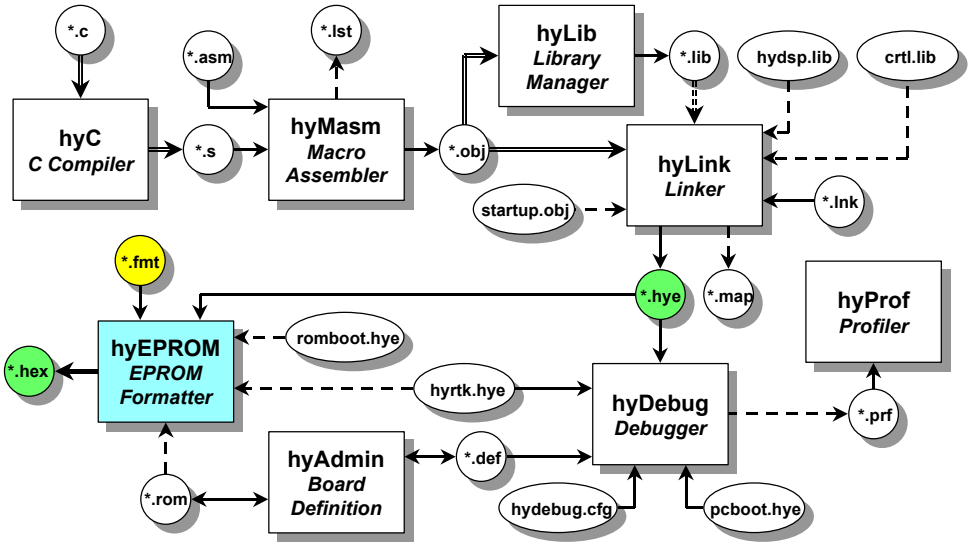
expression specifies the beginning address of the segment.

hyperstone electronics

Software Tools

hyEPROM

Unifying RISC and DSP



74

hyperstone electronics

EPROM Formatter hyEPROM

Unifying RISC and DSP

- produces one or more binary file(s) for programming EPROMs
- Command line invocation

  `hyeprom @commandfilename.fmt`

- one, two or four binary output files are generated according to the memory organization of the EPROMs.
- File: example1.fmt

```
OUTPUT      = example1.hex    ; output binary file
USER        = example1.hye    ; filename of user program
EPROMSIZE   = 128K            ; size of EPROM in Bytes
EPROMWIDTH  = 8               ; organization of EPROM
MEMBUSWIDTH = 8               ; bus size for accessing EPROM
BASEADDR    = $FFFE0000       ; 1 0000 0000_16 - 128KByte
```

75

The above formatter command file `example1.fmt` contains the following commands:

`OUTPUT =` `filename` denotes the name of the output binary file(s) used for programming the EPROMs. In case of more than one output file a digit is appended to the filename extension where the lower value indicates the lower address.

`USER =` `filename` denotes the name of the user program file

`EPROMSIZE` specifies the size of each EPROM output file in bytes. The optional suffix **`M`** means megabytes and **`K`** means kilobytes.

`EPROMWIDTH` specifies the organization of the EPROM-chip(s) (x8, x16 or x32 bit organization).

`MEMBUSWIDTH` specifies the bus size (8, 16 or 32 bits) for accessing the EPROM(s). The number of output files generated is calculated by the formula **`MEMBUSWIDTH / EPROMWIDTH`**.

Example: Accessing two EPROMs with 16 bit organization via a 32-bit memory bus: **`EPROMWIDTH`** = 16, **`MEMBUSWIDTH`** = 32

`MEMBUSWIDTH` / **`EPROMWIDTH`** = 2, therefore two EPROM files are generated.

`BASEADDR` specifies the base address of the EPROM. The default value is hexadecimal address ($100000000 - EPROM size), that is the EPROM is assumed at the end of memory area MEM3. For EPROMs in memory area MEM2, **`BASEADDR`** is usually hexadecimal address $80000000.

hyperstone electronics

Real-Time Operating System hyRTK

Unifying RISC and DSP

- Real-Time Operating System hyRTK
  - Stack-Level Tasks
  - Interrupt-Level Tasks
  - CreateTask
  - System Calls for Delaying Tasks
  - Guards
  - System Calls for accessing System Resources

hyperstone electronics

Real-Time Operating System hyRTK (1)

Unifying RISC and DSP

- multitasking
- up to 255 stack-level tasks with unique priority (highest priority: 0)
  only tasks with priority < 32 start running when scheduled
- up to 253 interrupt-level tasks with unique priority (highest priority: 0)
- pre-emptive, not time-sliced
  lower-priority tasks are preempted automatically by the highest-priority scheduled task
- task synchronization via Guards
- timing functions
- current size
  32 Kbytes (8000_{16} Bytes)
  executed in MEM0, address range $0000\ 0000_{16}$ - $0000\ 7FFF_{16}$
  or
  executed in MEM1, address range $4000\ 0000_{16}$ - $4000\ 7FFF_{16}$

Real-Time Operating System hyRTK (2)

Unifying RISC and DSP

q **Stack-Level Tasks**

- **own task control block (TCB) for each task**
- **defining Stack-Level Task in C with macro**

```
StackLevelTCB(TCBVariable, Priority,
              OnCreate, OnError, OnReset,
              SizeHardwareStack, SizeAggregateStack);
```

Macro `StackLevelTCB` declares the variable `TCBVariable` of type `StackLevelTCBType`

`Priority`	is the priority of the stack-level task each stack-level task must have a different priority when its priority is set in the range 0..31
`OnCreate`	points to the user defined task function (task entry point)
`OnError`	points to the user defined error function, optionally NULL
`OnReset`	points to the user defined reset function, optionally NULL
`SizeHardwareStack`	defines the size of the hardware stack in bytes
`SizeAggregateStack`	defines the size of the aggregate stack in bytes

78

Each stack-level task has a task control block (TCB). A TCB is declared by a macro; the macro provides initialization parameters and defines the size of the TCB. In the present version, a stack-level TCB has a size of 200 bytes.

The TCB macro for a stack-level task is applied in C as:

```
StackLevelTCB(Label, Priority, OnCreate, OnError, OnReset,
              SizeHardwareStack, SizeAggregateStack);
```

The C compiler treats the **`StackLevelTCB`** macro as a declaration of the variable `Label` with the predefined structure type **`StackLevelTCBType`**. The meaning and use of the parameters is the same as applied in assembler. The **`StackLevelTCB`** macro applied in C must not be placed in a function (e.g. **`main()`**), because the variable declared by the macro must be global.

A stack-level TCB which is declared in a C source module, can be imported in other C source modules as follows:

```
extern StackLevelTCBType Label;
```

Note: For the **`main()`** task, `OnError` is initialized to point to the C-function `raise()`. Optionally, `OnError` in other stack-level tasks may also be specified to point to `raise()`.

hyperstone electronics

Real-Time Operating System hyRTK (3)

Unifying RISC and DSP

q **Interrupt-Level Tasks**

- **own task control block (TCB) for each task**
- **defining Interrupt-Level Task in C with macro**

```
InterruptLevelTCB(TCBVariable, Priority, OnInterrupt, OnError, OnReset);
```

Macro `InterruptLevelTCB` declares the variable `TCBVariable` of type `InterruptLevelTCBType`

`Priority`	**is the priority of the interrupt-level task** `Priority` = 5 corresponds to pin IO3 `Priority` = 7 corresponds to pin INT1 `Priority` = 9 corresponds to pin INT2 `Priority` = 11 corresponds to pin INT3 `Priority` = 13 corresponds to pin INT4 `Priority` = 14 corresponds to pin IO1 `Priority` = 15 corresponds to pin IO2
`OnInterrupt`	**points to the user defined interrupt service function**
`OnError`	**points to the user defined error function, optionally NULL**
`OnReset`	**points to the user defined reset function, optionally NULL**

79

Each interrupt-level task has a task control block (TCB). A TCB is declared by a macro; the macro provides initialization parameters and defines the size of the TCB. In the present version, a interrupt-level TCB has a size of 80 bytes.

The TCB macro for a interrupt-level task is applied in C as:

```
InterruptLevelTCB(Label, Priority,
                  OnInterrupt, OnError, OnReset);
```

The C compiler treats the **`InterruptLevelTCB`** macro as a declaration of the variable `Label` with the predefined structure type **`InterruptLevelTCBType`**. The meaning and use of the parameters is the same as applied in assembler. The **`InterruptLevelTCB`** macro applied in C must not be placed in a function (e.g. **`main()`**), because the variable declared by the macro must be global.

A interrupt-level TCB which is declared in a C source module, can be imported in other C source modules as follows:

```
extern InterruptLevelTCBType Label;
```

An interrupt-level task may interrupt any stack-level task. Since interrupt-level tasks use the aggregate stack of the interrupted stack-level task, it must be guaranteed that each and every stack-level task provides enough aggregate stack space to fulfill the interrupt-level task's need for aggregate stack additionally to its own need. The hyC C compiler generates code so that an interrupt-level task written in C uses the aggregate stack of the interrupted stack-level task.

Real-Time Operating System hyRTK (4)

Unifying RISC and DSP

q Interrupt-Level Tasks and Interrupts

- interrupts start execution of an interrupt-level task
- macro `SetOwnTaskPointer(Label)`

  must be placed as the first statement of the entered interrupt function

  `Label` is the variable name of the TCB of the entered interrupt function

```
void MyInterrupt(void);    /* function prototype of interrupt function */
InterruptLevelTCB(MyInterruptTCB, 14, MyInterrupt, NULL, NULL);

void MyInterrupt(void) /* implement. of interrupt function */
{
  SetOwnTaskPointer(&MyInterruptTCB);
  ...
}
```

- interrupt-level tasks may interrupt any stack-level task
- interrupt-level tasks may not be interrupted themselves

Real-Time Operating System hyRTK (5)

- **CreateTask**

  Synopsis

```
#include <sys/hyrtk.h>
void CreateTask(StackLevelTCBType* TCBPointer);
void CreateTask(InterruptLevelTCBType* TCBPointer);
```

  Description

  The `CreateTask` function creates a task with the TCB addressed by `TCBPointer`.

  For stack-level tasks, a hardware stack and an aggregate stack of the specified size is allocated. `CreateTask` is invoked automatically for the first user task `main()`.

  For interrupt-level tasks, the interrupt entry is inserted in the trap entry table.

  Returns

  The `CreateTask` function returns no value.

hyperstone electronics

Real-Time Operating System hyRTK (6)

Unifying RISC and DSP

- **System Initialization creates 4 Tasks**
  - **stack-level task**
    - **SysTask (priority = 0)**
      - **handles communication between hyperstone system and host system when debugger is connected**
  - **interrupt-level tasks**
    - **TimerTask (priority = 254)**
      - **handles overflow of the 32-bit hardware timer into high-order word of Time**
    - **SysDriverTask (priority = 14 or priority = 9)**
      - **handles interrupt of UART (INT4)**
      - **dual-ported RAM (INT4 or INT2)**
    - **ProfilerTask (priority = 255)**
      - **handles optional profiling of user programs when profiler is active**

hyperstone electronics

Real-Time Operating System hyRTK (7)

Unifying RISC and DSP

- **GetHyTime**

  Synopsis

  ```
  #include <sys/hyrtk.h>
  signed long long int GetHyTime(void);
  ```

  Description

  Since system time is a value in the range of $0..2^{63}-1$, the date is also included and can be extracted.

  Returns

  The `GetHyTime` function returns the current system time in multiples of 1 microsecond.

Real-Time Operating System hyRTK (8)

- **DelayUntil**

  Synopsis

  ```
  #include <sys/hyrtk.h>
  void DelayUntil(signed long long int EventTime);
  ```

  Description

  The `DelayUntil` function suspends the calling function until the `EventTime` is reached.

  When the `EventTime` is less than the current system time, the current system time is placed in the `EventTime`.

  The highest-priority scheduled stack-level task starts running.

  Returns

  The `DelayUntil` function returns no value.

Real-Time Operating System hyRTK (9)

- **DelayBy**

  Synopsis

  ```
  #include <sys/hyrtk.h>
  void DelayBy(signed long int TimeUnits);
  ```

  Description

  The `DelayBy` function suspends the calling function for `TimeUnits`.

  A time unit represents 1 microsecond.

  A negative value of `TimeUnits` is treated as 0.

  The highest-priority scheduled stack-level task starts running.

  Returns

  The `DelayBy` function returns no value.

hyperstone electronics

Real-Time Operating System hyRTK (10)

Unifying RISC and DSP

- **Guards**
  - basic means to synchronize tasks
  - only guard state = 0 lets a task pass and execute beyond a
    `WaitGuard()`
    `WaitGuardMax()`
  - guard state <> 0 sets any task which tries to pass via
    `WaitGuard()`
    `WaitGuardMax()`
    to the task state waiting
  - defining Guards in C with macro
    `Guard(GuardVariable, InitialGuardState);`
    Macro `Guard` declares the variable `GuardVariable` with the predefined structure `GuardType`
    `InitialGuardState` can be any C integer constant expression

86

Real-Time Operating System hyRTK (11)

- **ClearGuard**

  Synopsis

  ```
  #include <sys/hyrtk.h>
  void ClearGuard(GuardType* GuardPointer);
  ```

  Description

  The `ClearGuard` function sets the guard denoted by `GuardPointer` to 0.

  An interrupt-level task which calls `ClearGuard` must return with either a `ClearGuardReturn()`, `ClearGuardBitsReturn()`, `DelayByReturn()` or `DelayUntilReturn()` function call.

  Returns

  The `ClearGuard` function returns no value.

Real-Time Operating System hyRTK (12)

- **SetGuard**

  Synopsis

  ```
  #include <sys/hyrtk.h>
  void SetGuard(GuardType* GuardPointer);
  ```

  Description

  The `SetGuard` function sets the guard denoted by `GuardPointer` to 1.

  Returns

  The `SetGuard` function returns no value.

hyperstone electronics

Real-Time Operating System hyRTK (13)

Unifying RISC and DSP

- **WaitGuard**

  Synopsis

  ```
  #include <sys/hyrtk.h>
  void WaitGuard(GuardType* GuardPointer);
  ```

  Description

  When the guard denoted by `GuardPointer` is 0, execution of the calling task proceeds. Afterwards the referenced guard is set to 1.

  When the guard denoted by `GuardPointer` is <> 0, the calling task is suspended and set waiting on the addressed guard. The highest-priority scheduled stack-level task starts running.

  Returns

  The `WaitGuard` function returns no value.

Real-Time Operating System hyRTK (14)

- **WaitGuardMax**

Synopsis

```
#include <sys/hyrtk.h>
void WaitGuardMax(GuardType* GuardPointer,
                  signed long int TimeUnits);
```

Description

When the guard denoted by `GuardPointer` is 0, execution of the calling task proceeds. Afterwards the referenced guard is set to 1.

When the guard denoted by `GuardPointer` is <> 0, the calling task is suspended and set waiting on the addressed guard. The highest-priority scheduled stack-level task starts running.

When `TimeUnits` passed and the task is still waiting, a timeout occurs. Then the task state of the calling state is set from waiting to scheduled and the referenced guard is set to 1.

Returns

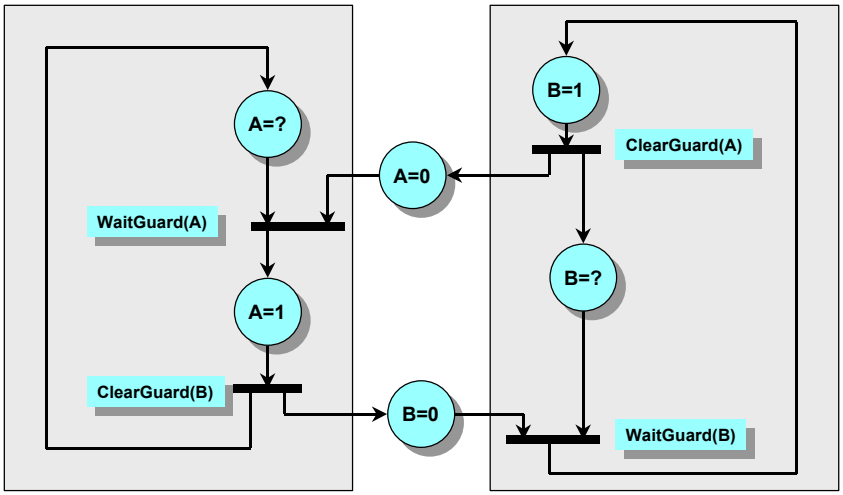
The `WaitGuardMax` function returns no value.

hyperstone electronics

Real-Time Operating System hyRTK (15)

Unifying RISC and DSP

- **WaitGuard() and ClearGuard()**



Stack-Level Task 1 **Stack-Level Task 2**

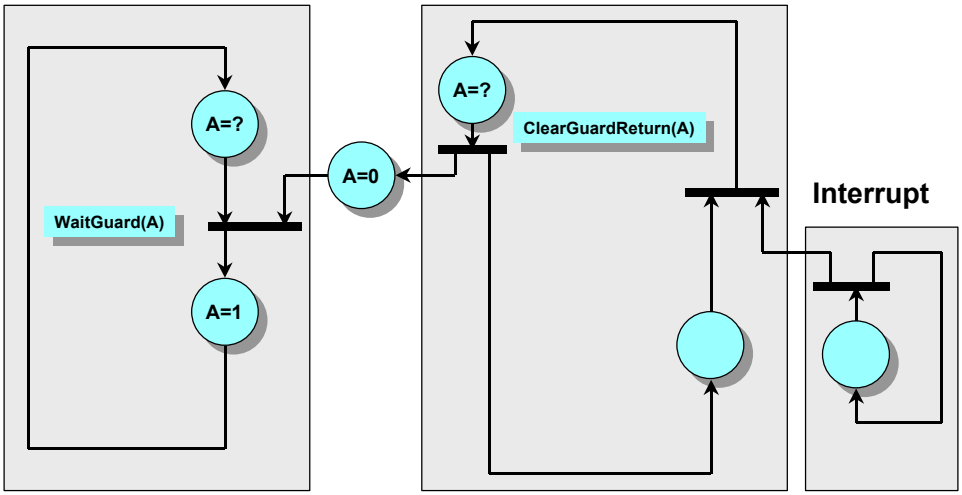
91

hyperstone electronics

Real-Time Operating System hyRTK (16)

Unifying RISC and DSP

- WaitGuard() and ClearGuardReturn()



Stack-Level Task

Interrupt-Level Task

92

Real-Time Operating System hyRTK (17)

Unifying RISC and DSP

- **GetBCR**

  Synopsis

  ```
  #include <sys/hyrtk.h>
  unsigned long int GetBCR(void);
  ```

  Returns

  The `GetBCR` function returns the current value of the Bus Control Register BCR.

- **GetMCR**

  Synopsis

  ```
  #include <sys/hyrtk.h>
  unsigned long int GetMCR(void);
  ```

  Returns

  The `GetMCR` function returns the current value of the Memory Control Register MCR.

Real-Time Operating System hyRTK (18)

- **UpdateBCR**

Synopsis

```
#include <sys/hyrtk.h>
void UpdateBCR(unsigned long int SetBits
               unsigned long int ClearBits);
```

Description

The `UpdateBCR` function modifies the value of the bus control register BCR.

Each bit set in `SetBits` sets the corresponding bit in the BCR.

Each bit set in `ClearBits` clears the corresponding bit in the BCR.

If the same bit is set in `SetBits` and `ClearBits`, the corresponding bit in the BCR will be inverted (toggled).

Returns

The `UpdateBCR` function returns no value.

Real-Time Operating System hyRTK (19)

- **UpdateMCR**

Synopsis

```
#include <sys/hyrtk.h>
void UpdateMCR(unsigned long int SetBits
               unsigned long int ClearBits);
```

Description

The `UpdateMCR` function modifies the value of the memory control register MCR.

Each bit set in `SetBits` sets the corresponding bit in the MCR.

Each bit set in `ClearBits` clears the corresponding bit in the MCR.

If the same bit is set in `SetBits` and `ClearBits`, the corresponding bit in the MCR will be inverted (toggled).

Returns

The `UpdateMCR` function returns no value.

Real-Time Operating System hyRTK (20)

- **GetISR**

  Synopsis

  ```
  #include <sys/hyrtk.h>
  unsigned long int GetISR(void);
  ```

  Returns

  The `GetISR` function returns the current value of the Input Status Register ISR.

- **GetFCR**

  Synopsis

  ```
  #include <sys/hyrtk.h>
  unsigned long int GetFCR(void);
  ```

  Returns

  The `GetFCR` function returns the current value of the Function Control Register FCR.

Real-Time Operating System hyRTK (21)

- **UpdateFCR**

Synopsis

```
#include <sys/hyrtk.h>
void UpdateFCR(unsigned long int SetBits
               unsigned long int ClearBits);
```

Description

The `UpdateFCR` function modifies the value of the function control register FCR.

Each bit set in `SetBits` sets the corresponding bit in the FCR.

Each bit set in `ClearBits` clears the corresponding bit in the FCR.

If the same bit is set in `SetBits` and `ClearBits`, the corresponding bit in the FCR will be inverted (toggled).

Returns

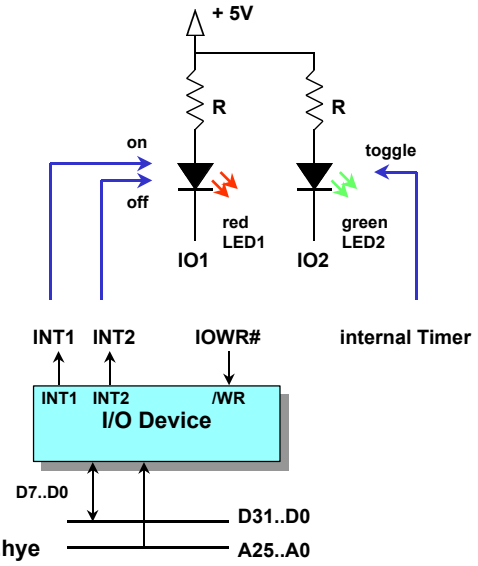
The `UpdateFCR` function returns no value.

hyperstone electronics

C Example (with Real-Time Operating System)

Unifying RISC and DSP

- 2 LED s
  - IO1 drives red LED1
  - IO2 drives green LED2
- I/O Device
  - INT1 switches red LED1 on
  - INT2 switches red LED1 off
  - Interrupt INT1 and INT2 can be cleared via write access
- Internal Timer
  - toggles green LED2 each 1 second
- Used Software Development Tools
  - C Compiler hyC
  - C Run-Time Library crtl.lib
  - Real-Time Operating System hyRTK
  - Linker hyLink
  - Board Definition hyAdmin
  - Debugger hyDebug
  - Boot Loader romboot.hye and pcboot.hye
  - EPROM Formatter hyEPROM



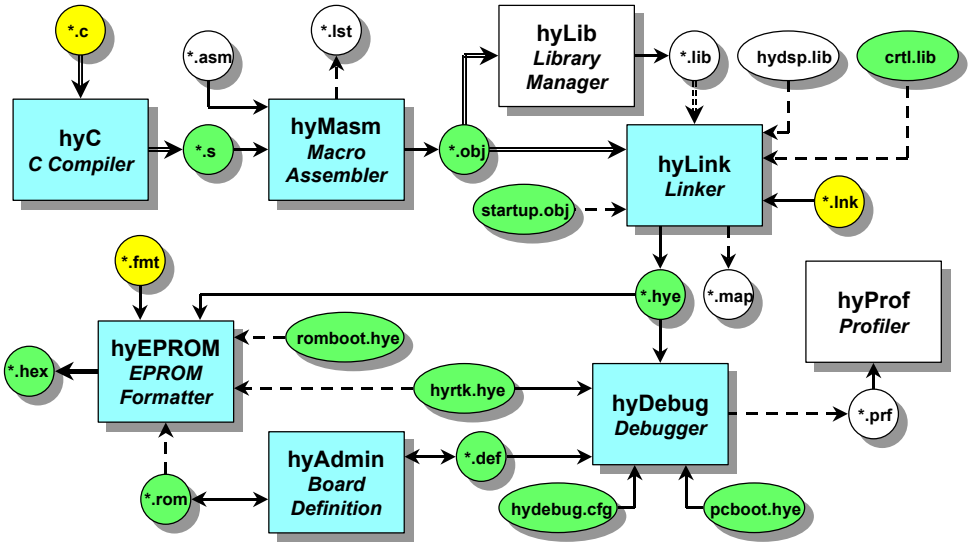
98

hyperstone electronics

Software Tools

hyC, hyMasm, hyLink, hyAdmin, hyDebug, hyEPROM

Unifying RISC and DSP



99

hyperstone electronics

main.c

Unifying RISC and DSP

```
#include <sys\hyrtk.h>
#include "int.h"
#include "global.h"

extern InterruptLevelTCBType InterruptTCB1;
extern InterruptLevelTCBType InterruptTCB2;

void main(void)
{
  CreateTask(&InterruptTCB1);
  CreateTask(&InterruptTCB2);

  /* set IO1 and IO2 to output mode, enable INT1 and INT2 */
  UpdateFCR(IO1Mask | IO2Mask,  IO1Direction | IO2Direction | INT1Mask | INT2Mask);
  while (1)
  {
    /* toggle green LED2 */
    UpdateFCR(IO2Polarity, IO2Polarity);
    DelayBy(1000000);  /* delay 1 second */
  }
}
```

hyperstone electronics

global.h

Unifying RISC and DSP

```
#ifndef _global_h
#define _global_h

#define INT1Mask  1<<28  /* 0 = enable */
#define INT2Mask  1<<29  /* 0 = enable */

/* red LED */
#define IO1Direction 1<<2 /* 0 = Output          */
#define IO1Polarity  1<<1 /* 1 = Non-Inverted    */
#define IO1Mask      1<<0 /* must be 1 on Output */

/* green LED */
#define IO2Direction 1<<6 /* 0 = Output           */
#define IO2Polarity  1<<5 /* 1 = Non-Inverted     */
#define IO2Mask      1<<4 /* must be 1 on Output  */

/* I/O Peripheral */
#define PeripheralAddr 0x03FFF7F8 /* specify according to the connected hardware */
#define ClearINT1      0xF        /* value to clear INT1 */
#define ClearINT2      0x0        /* value to clear INT2 */

#endif
```

hyperstone electronics

int.h

Unifying RISC and DSP

```
#ifndef _int_h
#define _int_h

/* Function Prototypes */
void InterruptFunction1(void);
void InterruptFunction2(void);

#endif
```

hyperstone electronics

int.c

Unifying RISC and DSP

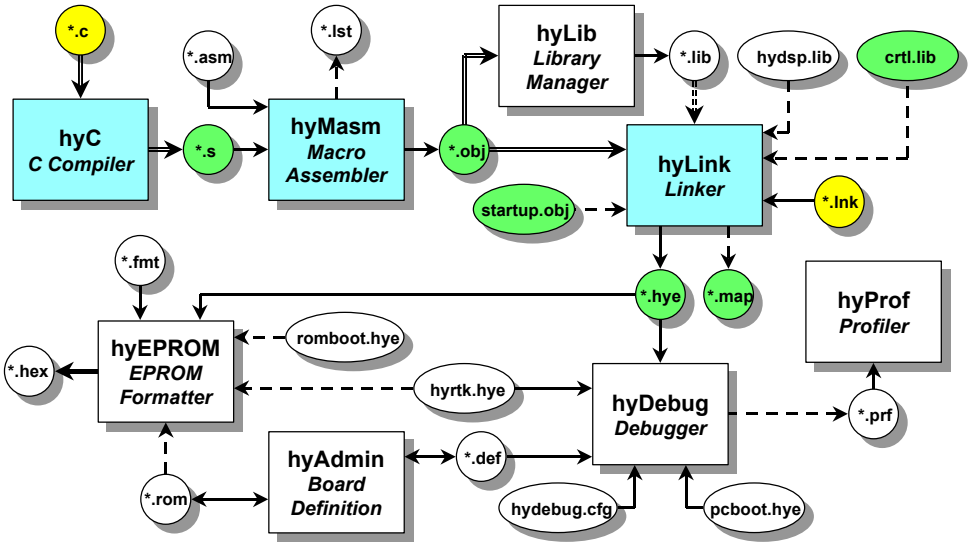
```
#include <sys\hyrtk.h>
#include <io.h>
#include "global.h"
#include "int.h"

InterruptLevelTCB(InterruptTCB1, 7, InterruptFunction1, NULL, NULL);
InterruptLevelTCB(InterruptTCB2, 9, InterruptFunction2, NULL, NULL);

void InterruptFunction1(void)
{
  SetOwnTaskPointer(&InterruptTCB1);
  outpw(PeripheralAddr, ClearINT1);
  UpdateFCR(0, IO1Polarity);
}

void InterruptFunction2(void)
{
  SetOwnTaskPointer(&InterruptTCB2);
  outpw(PeripheralAddr, ClearINT2);
  UpdateFCR(IO1Polarity, 0);
}
```

hyperstone electronics

Software Tools

hyC

Unifying RISC and DSP



104

hyperstone electronics

ANSI C Compiler hyC (1)

Unifying RISC and DSP

- translates ANSI C source files into assemby language source programs
- assembly language files can be automatically passed to the assembler hyMasm to generate object modules
- object modules can be automatically passed to the linker hyLink to generate a *hyper*stone executable file
- Command line invocation
  `hyc {options} filename {filename}`
- Options
  `-S`
  compiles C source files into assembler code, but does not invoke the assembler
  `-c`
  compiles and assembles C source files and assembly language source programs to object files, but does not link
  `-$LinkerCommandFile`
  uses the linker command file `LinkerCommandFile` for linking the object files, when neither `-S` option nor `-c` option specified, `-$` option must be specified

105

The hyC *hyper***stone** ANSI C compiler takes a C source file and translates it to assembler statements which may then be automatically passed to the assembler in order to get an objet file.

The hyC compiler may be invoked by entering **hyc** followed by options and one ore more C source file names. The syntax for the entire command line is as follows:

hyc `{option} filename {filename}`

Options must be preceded by a minus sign (**-**) and must be separated by at least one space character; multiple single-letter options may *not* be grouped: **-dr** is different from **-d -r**.

`filename` denote the filenames of the source file(s) to be compiled. File names which end in **.c** are taken as C source to be pre-processed, compiled and assembled. Please note that the **.c** file extension is *not* optional. If the **.c** file extension is omitted, an error will be flagged. Multiple filenames can be specified on the command line but filenames containing wildcards, such as ***.c** or **xyz?.c**, are not allowed.

Compiler output files which end in **.s** and assembler source files with the extension **.asm** are assembled when specified on the command line.

When you invoke the C compiler with a C source file, it automatically does pre-processing, compilation, assembly and linking (when specifying the **-$** option). The output is then an executable program with the file extension **.hye** which is ready for loading to the target system. Command options allow halting the compilation process after a certain stage of processing.

ANSI C Compiler hyC (2)

- Options

  `-g`

  produces debugging information for source-level debugging in combination with the debugger hyDebug

  `-O`

  with `-O` (upper-case O), the compiler tries to reduce code size and execution time

  `-w`

  inhibits all warning messages

  `-Wall`

  prints all warning messages

  `-DSymbol[=IntVal]`

  defines the symbol `Symbol` together with optional value `IntVal`

hyperstone electronics

ANSI C Compiler hyC (3)

Unifying RISC and DSP

q **Object Sizes**

Data Type	Size	Alignment
unsigned char	8 bits	byte boundary
signed char	8 bits	
unsigned short int	16 bits	halfword boundary
signed short int	16 bits	max. 1 byte of padding
unsigned int	32 bits	word boundary
signed int	32 bits	max. 3 bytes of padding
unsigned long int	32 bits	
signed long int	32 bits	
unsigned long long int	64 bits	
signed long long int	64 bits	
float	32 bits	
double	64 bits	
pointer	32 bits	

hyperstone electronics

ANSI C Compiler hyC (4)

Unifying RISC and DSP

q **Arguments Passing**

- **only 6 local registers may be used to pass arguments to a function**
- **all other arguments and structure arguments must be stored on the aggregate stack**
- **register stack arguments:**

```
<Caller>:
MOV     Ln,   <arg0>
MOV     Ln+1, <argn1>
          :
MOV     Ln+5, <arg5>
CALL    Ln+6, <Callee>

<Callee>:
FRAME   Ld, Ls     ; Ld is the largest register used + 1
          :        ; Ls is the number of register arguments passed
RET     PC, Ls
```

108

Only 6 registers (as seen by the called subprogram) may be used to pass arguments to a function. This reduces the probability of spilling arguments. All other arguments must be stored on the aggregate stack. Structure arguments are always stored on the aggregate stack.

hyperstone electronics

ANSI C Compiler hyC (6)

Unifying RISC and DSP

q **Return Values**

- **function return value in L0:**
  char, short int, int, long int, float and pointers
- **function return value in L0 and L1:**
  double and long long int
- **If a function returns two words (double or long long int) and takes only one word argument, an extra register must be preserved by the caller to hold the result:**

```
MOVI    L4, $1000       ; pass $1000 as arg1
CALL    L6, func1       ; skip L5
MOVD    L0, L4          ; result is in L4//L5
```

109

Function results are always returned in L0 for **`char`**, **`short int`**, **`int`**, **`long int`**, **`float`** and pointers, while **`double`** and **`long long int`** are returned in L0 and L1.

Note that if a function returns a value, then enough registers must be allocated to hold the result.

ANSI C Compiler hyC (7)

- Names of Global Variables and Functions
  - prefixed with an underscore

Declaration	Compiler Name
`int i;`	`_i`
`int int_num;`	`_int_num`
`void funct(void);`	`_funct`

- Names of Local Static Variables
  - prefixed with an underscore and a numerical value

Declaration	Compiler Name
`void func_a(void)`	
`{`	
`static int i;`	`_0i`
`static int_num;`	`_1int_num`
`}`	
`void func_b(void)`	
`{`	
`static int i;`	`_2i`
`static int_num;`	`_3int_num`
`}`	

ANSI C Compiler hyC (8)

Unifying RISC and DSP

- **Default Segment Names**

Segment Name	Usage	Example
`code`	program code	if (A != B) C = 0;
`text`	static initialized data	int a[10] = {0}, b=1;
`far_bss`	static uninitialized data	int a[10], b;

- **Renaming Predefined Segment Names with Compiler Option**

  `-mseg-code=name`

  renames segment code to segment `name`

  `-mseg-text=name`

  renames segment `text` **to segment** `name`

  `-mseg-far_bss=name`

  renames segment `far_bss` **to segment** `name`

- **Creating Additionally Segment Names with Compiler Option**

  `-mscalar-seg`

  generates additionally segments `scalar_text` **and** `scalar_far_bss`

hyperstone electronics

ANSI C Compiler hyC (9)

Unifying RISC and DSP

- **Linker Command File (used by hyLink) and C Programs**
  - **link with start-up code startup.obj and C run-time library crtl.lib**
  - **priority of task main**
  - **size of hardware stack of task main**
  - **size of aggregate stack of task main**
  - **compile with hyC and option `-$LinkerCommandFile` or link with hyLink and `@LinkerCommandFile`**

112

After all C and assembler modules have been compiled, they are ready to link together with the **`startup.obj`** start-up code and the **`crtl.lib`** run-time library to a single executable program.

Modules can either separately compiled and linked later by invoking the linker in a separate pass or the C compiler can be directed to invoke the linker automatically by specifying the **`-$`** option. In any case a linker command file containing the names of all modules making up the executable file and some commands controlling the linkage process has to be set up.

The **`ORDER`** command tells the linker to locate the segments **`code`**, **`text`** and **`far_bss`** in consecutive ascending order.

The **`LOCATE`** command specifies the start address of the first segment to begin at 8000_{16}.

Please note that addresses 0000_{16} - $7FFF_{16}$ or 4000 0000_{16} - 4000 $7FFF_{16}$, respectively are reserved for the real-time operating system hyRTK.

The priority of the program **`Priority`**, the size of the hardware stack defined by **`Stack1Size`** and the size of the aggregate stack **`Stack2Size`** have to be specified.

When calculating the size of the hardware stack **`Stack1Size`** and the size of the aggregate stack **`Stack2Size`**, please keep in mind that recursive functions need a large amount of stack space depending on the depth of recursion and that the C run-time library needs approximately 512 bytes for the hardware stack and aggregate stack.

hyperstone electronics

main.lnk (1)

Unifying RISC and DSP

```
; linker command file for C example

; RENAME Command
; rename segment code and text to IRAMcode and IRAMtext
RENAME int.o code = IRAMcode
RENAME int.o text = IRAMtext

; LINK Command
main.hye = main.o, int.o, startup.obj crtl.lib

; ORDER Command
; Order segments IRAMcode, IRAMtext
ORDER  IRAMcode, IRAMtext

; LOCATE Command
; Locate IRAMcode and IRAMtext in IRAM
LOCATE IRAMcode at $C0000000
```

hyperstone electronics

main.lnk (2)

Unifying RISC and DSP

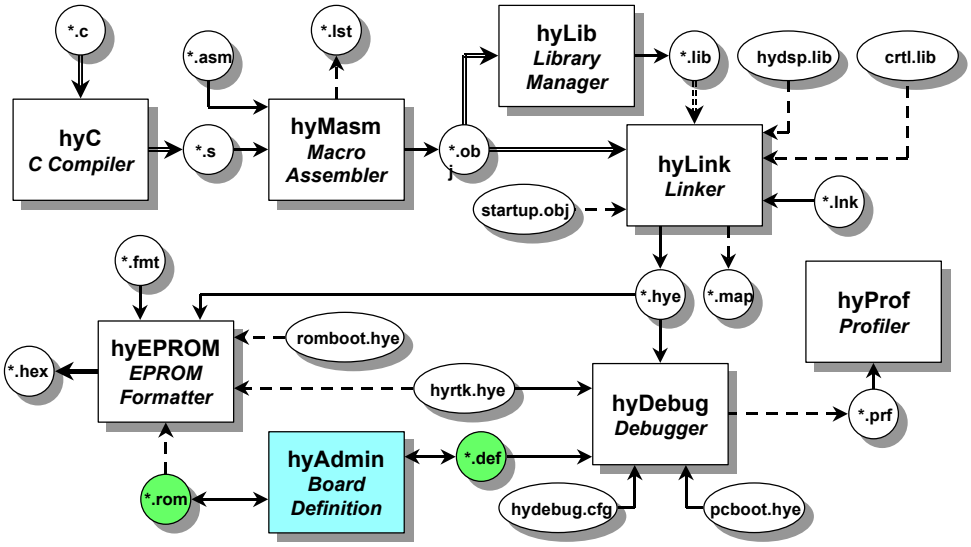
```
; ORDER Command
; Order segments code, text and far_bss
ORDER  code, text, far_bss

; LOCATE COMMAND
; locate segments code, text, far_bss at address $8000
LOCATE code at $8000

; Define program priority and stack sizes
DEFINE Priority = 31
DEFINE Stack1Size = 2048      ; size of hardware stack
DEFINE Stack2Size = 2048      ; size of aggregate stack
```

hyperstone electronics

Software Tools

hyAdmin

Unifying RISC and DSP



115

Board Administration Program hyAdmin (1)

Unifying RISC and DSP

- provides service for generating and editing board definition files (e.g.) boarddef.def and boarddef.rom
- these files contains settings of up to 255 different board configurations
- a board configuration comprise following settings:

BCR	Bus Control Register
MCR	Memory Control Register
TPR	Timer Prescaler Register

- each board configuration can be identified with its bord type number (1..255)
- function control register FCR cannot be set via board definition file
- Following tools use the board definition file to initialized the BCR, MCR and TPR according to the hardware environment:
  - pcboot.hye in use with the debugger hyDebug
  - romboot.hye in use with EPROM formatter hyEPROM

116

The board definition program hyAdmin is invoked with the following command line:

```
hyadmin filename
```

where filename denotes the board definition file (usually **`boarddef.def`**). If you specify a non-existing file, hyAdmin automatically prompts you to create a new file. In this case type **`y`** or **`Y`** to create it or **`n`** or **`N`** to terminate hyAdmin.

If the file filename exists and is a valid board definition file, hyAdmin automatically loads this file for editing.

Board Administration Program hyAdmin (2)

Unifying RISC and DSP

```
Hyperstone Board Administration Program Version 2.04, Copyright(c) 1996-1997
Board Type 2          BCR F37505CB      TPR 00300000
CPU Type   E1-32      MCR FDD980F0

MEM0 Memory Type       DRAM                MEM2 Bus Size        8 bit
MEM0 Bus Size          32 bit              MEM2 Parity Check    Disabled
MEM0 Parity Check      Disabled            MEM2 Setup Time      0 clock cycles
MEM0 Access Time       2 clock cycles      MEM2 Access Time     8 clock cycles
MEM0 RAS Precharge     2 clock cycles      MEM2 Bus Hold Time   3 clock cycles
MEM0 RAS To CAS Delay  2 clock cycles      MEM2 Bus Hold Break  Enabled
MEM0 Refresh Rate      512 clock cycles
MEM0 Page Size Code    A11..A2

MEM1 Bus Size          32 bit              MEM3 Bus Size        8 bit
MEM1 Parity Check      Disabled            MEM3 Parity Check    Disabled
MEM1 Access Time       2 clock cycles      MEM3 Access Time     4 clock cycles
MEM1 Bus Hold Time     0 clock cycles      MEM3 Bus Hold Time   2 clock cycles
MEM1 Bus Hold Break    Enabled             MEM3 Bus Hold Break  Enabled

F1: Help        F4: Previous      F7: Copy        F9 : Save & Exit
F3: Delete      F5: Next          F8: Paste       F10: Exit
```

hyperstone electronics

Board Administration Program hyAdmin (3)

Unifying RISC and DSP

- **following predefined board types are kept in the board definition file boarddef.def and boarddef.rom:**

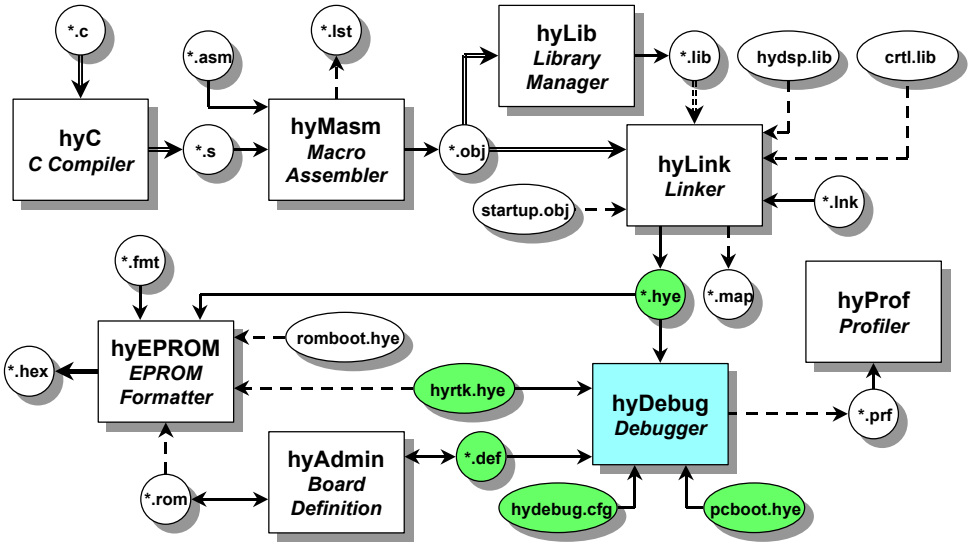
Board Type	Board Type
1	Development board (version 10/95) with *hyper***stone** E1-32N (version U2)
2	Development board (version 11/95) with *hyper***stone** E1-32N (version U3) @ 50 MHz
3	Reserved by *hyper***stone**
4	Development board (version 12/95) with *hyper***stone** E1-32N (version LL) @ 66 MHz
5	Development board (revision 5) with *hyper***stone** E1-32XN (version U4)
6	Reserved by *hyper***stone**
7	Optical character recognition board (version 08/95) with *hyper***stone** E1-32N (version U2)
8	Reserved by *hyper***stone**
9	Reserved by *hyper***stone**
10	Reserved by *hyper***stone**
11	Reserved by *hyper***stone**
12	Reserved by *hyper***stone**
13	Reserved by *hyper***stone**
14	Reserved by *hyper***stone**
15	Reserved by *hyper***stone**

hyperstone electronics

Software Tools

hyDebug

Unifying RISC and DSP



119

hyperstone electronics

hyDebug

Unifying RISC and DSP

```
 File  Search  Run  Profiler  Debug  Windows
                          Task States
SysTask          Scheduled   Running Prio:00  WaitFlag:00  No Error
Main             Scheduled   Stopped Prio:31  WaitFlag:00  No Error

[■]═══ D:\HYDEVE~1\HYMANUAL\HYTUTO~1\PROGRAMS\EXAMPLE\C\MAIN ═══1=[↑]
#include <sys\hyrtk.h>
#include "int.h"
#include "global.h"

extern InterruptLevelTCBType InterruptTCB1;
extern InterruptLevelTCBType InterruptTCB2;


void main(void)
{
  CreateTask(&InterruptTCB1);
  CreateTask(&InterruptTCB2);

  /* void UpdateFCR(unsigned long SetBits, unsigned long ClearBits) */
  UpdateFCR(IO1Mask | IO2Mask,
═══ 11:1 ═══
F2 Register  F3 Disass  F4 Goto cursor  F5 Run  F6 Next  F7 Trace  F8 Step
```

120

hyperstone electronics

Profiler hyProf

Unifying RISC and DSP

Profile Data

Function	Time	Percentage
_funcA	368.000msec	81.42%
[00014]	29.000msec	6.42%
[00017]	297.000msec	65.71%
[00018]	33.000msec	7.30%
[00019]	9.000msec	1.99%
_funcB	84.000msec	18.58%
[00025]	79.000msec	17.48%
[00026]	5.000msec	1.11%

Zoom Unzoom Cancel

```
void funcA(void)
{
  int i, a;

  for (i = 0; i < 10; i++)
    a = 0;
}
```

```
void funcB(void)
{
  int i;

  for (i = 0; i < 100000; i++)
    funcA();
}
```

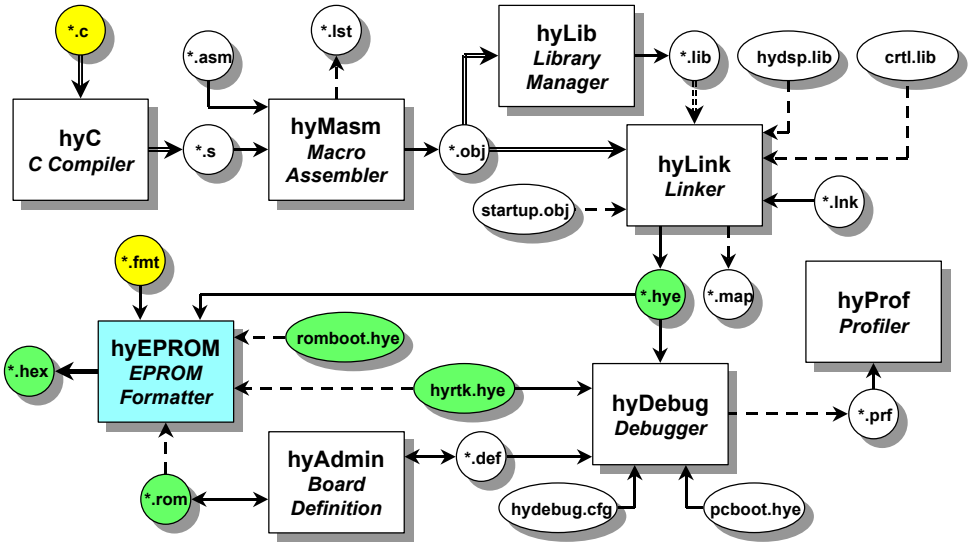
121

hyperstone electronics

Software Tools

hyEPROM

Unifying RISC and DSP



122

hyperstone electronics

main.fmt

Unifying RISC and DSP

```
OUTPUT        = EPROM.HEX                    ; Binary Output File
SYSTEM        = c:\hstone\bin\hyrtk.hye      ; Real-Time Operating System
BOOT          = c:\hstone\bin\romboot.hye    ; Bootloader
BOARDDEF      = c:\hstone\eprom\boarddef.rom ; Board Definition File
BOARDTYPE     = 2                            ; Hyperstone Development Board
USER          = main.hye                     ; User Program
EPROMSIZE     = 128K                         ; Size of EPROM in Bytes
EPROMWIDTH    = 8                            ; Data Bus Width of EPROM in Bits
MEMBUSWIDTH   = 8                            ; Bus Width of MEM3 in Bits
```

123

SYSTEM = `filename`
`filename` denotes the name of the operating system executable file (**hyrtk.hye**)

BOOT = `filename`
`filename` denotes the name of the boot file containing the boot loader (**romboot.hye**).

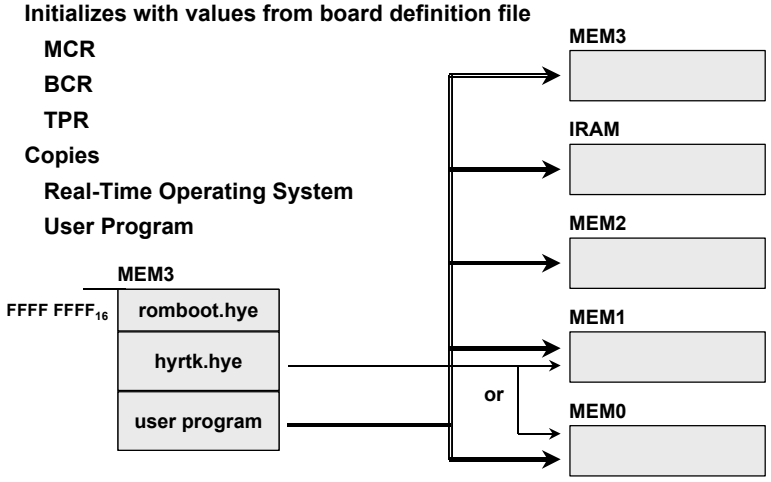
BOARDDEF = `filename`
`filename` denotes the name of the board definition file. The board definition file can contain up to 255 different board configurations. The board definition file can be generated and edited by the utility program hyADMIN.

BOARDTYPE
selects the desired board configuration (1..255) in the board definition file. A board configuration specifies the board clock frequency and the BCR and MCR register setting of the board.

Boot Loader romboot.hye and pcboot.hye

Unifying RISC and DSP

- Reset Handler
- Initializes with values from board definition file
  - MCR
  - BCR
  - TPR
- Copies
  - Real-Time Operating System
  - User Program



124

hyperstone electronics

DSP Unit

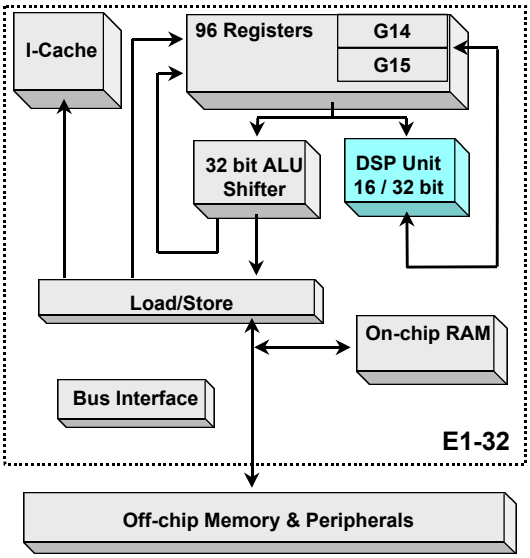
Unifying RISC and DSP

- DSP Unit
  - ALU and DSP Unit
  - Parallelism ALU - DSP
  - Example Dot Product

hyperstone electronics

ALU and DSP Unit

Unifying RISC and DSP



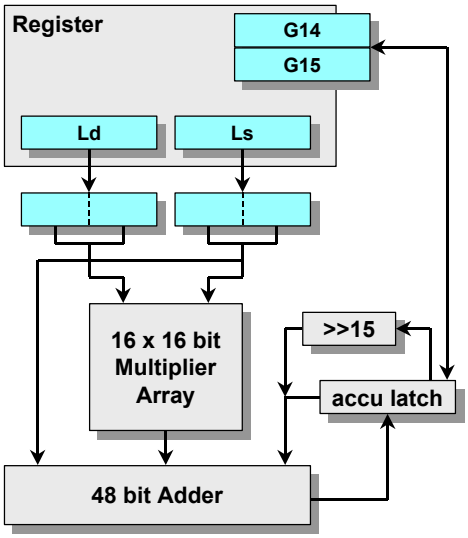
- Single issue
- Single-cycle ALU instr.
- 16 bit subword processing
- 16 / 32 bit DSP arithmetic
- Latency based parallelism of ALU - Ld/St - DSP
- Complex DSP arithmetic

126

hyperstone electronics

DSP Unit

Unifying RISC and DSP



Instruction		Remarks	
EMUL	Ld, Ls	G15	:= Ld · Ls (signed and unsigned)
EMULS	Ld, Ls	G14//G15	:= Ld · Ls (signed)
EMULU	Ld, Ls	G14//G15	:= Ld · Ls (unsigned)
EMAC	Ld, Ls	G15	:= G15 + Ld · Ls
EMACD	Ld, Ls	G14//G15	:= G14//G15 + Ld · Ls
EMSUB	Ld, Ls	G15	:= G15 – Ld · Ls
EMSUBD	Ld, Ls	G14//G15	:= G14//G15 – Ld · Ls
EHMAC	Ld, Ls	G15	:= G15 + LdH · LsH + LdL · LsL
EHMACD	Ld, Ls	G14//G15	:= G14//G15 + LdH · LsH + LdL · LsL
EHCMULD	Ld, Ls	G14	:= LdH · LsH – LdL · LsL
		G15	:= LdH · LsL + LdL · LsH
EHCMACD	Ld, Ls	G14	:= G14 + LdH · LsH – LdL · LsL
		G15	:= G15 + LdH · LsL + LdL · LsH
EHCSUMD	Ld, Ls	G14H	:= LdH + G14
		G14L	:= LdL + G15
		G15H	:= LdH – G14
		G15L	:= LdL – G15
EHCFFTD	Ld, Ls	G14H	:= LdH + G14>>15
		G14L	:= LdL + G15>>15
		G15H	:= LdH – G14>>15
		G15L	:= LdL – G15>>15

hyperstone electronics

Parallelism ALU - DSP

Unifying RISC and DSP

```
        LDD.P   L0, L5          : load data with postincrement
Label:  EMULU   L5, L6          ; multiply 32 bit x 32 bit
        LDD.P   L0, L5          : load new data with postincrement
        ADDI    L4, -1          ; decrement loop index
        DBGT    Label           ; conditional delayed branch
        STD.P   L1, G14         ; store G14//G15
```

Cycle	DSP Instruction	ALU Instruction	Load/Store Instructions
1	Issue		
2	Latency		Load
3	Latency		Load
4	Latency	ADD	Latency
5	Latency	Branch	Latency
			Store
			Store

hyperstone electronics

Dot Product Evaluation (1)

Unifying RISC and DSP

```
long int x[100], y[100];
long long int accu;

accu = 0;
for (i = 0; i < 100; i++)
  accu += x[i] * y[i];


; L0 = &x
; L1 = &y
   LDW.P        L0, L4          ; L4 = x[0]
   LDW.P        L1, L5          ; L5 = y[0]
   MOVD         G14, 0          ; clear accumulator
   MOVI         L6, 99          ; loop index, 99 down to 0, sets flags
loop:
   EMACD        L4, L5          ; multiply-accumulate
   LDW.P        L0, L4          ; load new x, postincrement address update
   LDW.P        L1, L5          ; load new y, postincrement address update
   DBGT         loop            ; delayed branch, if N or Z flags are 0
   ADDI         L6, -1          ; loop index update
```

129

The resources of the E1-32/E1-16 (ALU, Load/Store Pipeline and DSP Unit) are 90% used.

Dot Product Evaluation (2)

Unifying RISC and DSP

```
; L0= &x
; L1= &y
   LDW.P        L0, L4          ; L4 = x[0]
   LDW.P        L1, L5          ; L5 = y[0]
   MOVD         G14, 0          ; clear accumulator
   MOVI         L8, 100         ; loop index, 100 down to 1
loop:
   LDW.P        L0, L6          ; load new x, postincrement address update
   LDW.P        L1, L7          ; load new y, postincrement address update
   ADDI         L8, -2          ; loop index update
   EMACD        L4, L5          ; multiply-accumulate
   LDW.P        L0, L4          ; load new x, postincrement address update
   LDW.P        L1, L5          ; load new y, postincrement address update
   DBGT         loop            ; delayed branch, if N or Z flags are 0
   EMACD        L6, L7          ; multiply-accumulate
```

130

The resources of the E1-32/E1-16 (ALU, Load/Store Pipeline and DSP Unit) are 90% used.