Microprocessors

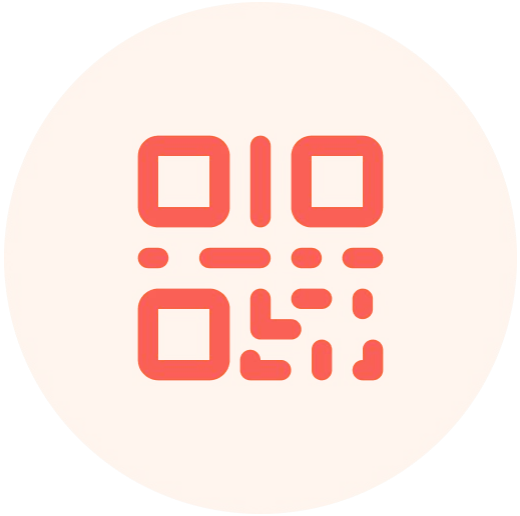
Tuba Ayhan

MEF University

I/O Devices

Computer Organization and Embedded Systems, Hamacher et. al

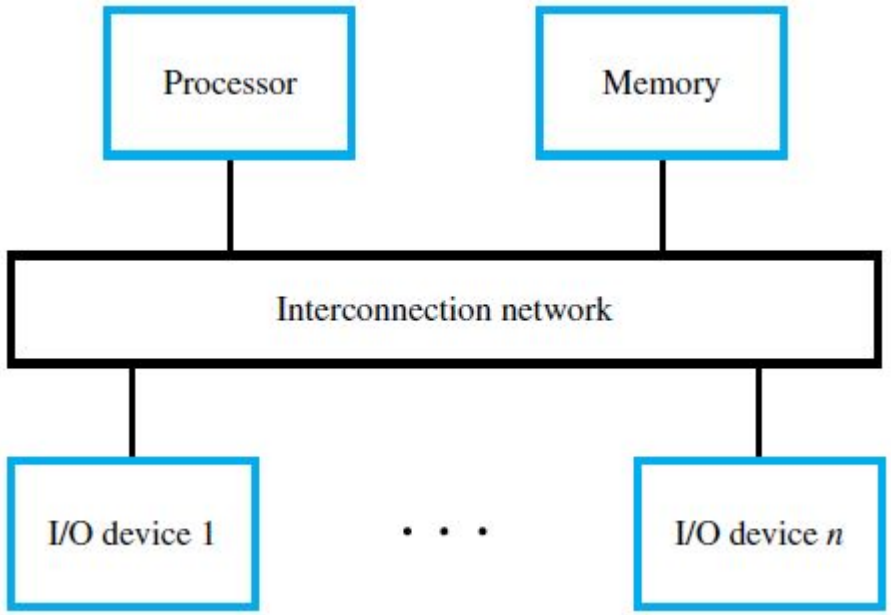
slido



Join at slido.com
#491634

ⓘ Start presenting to display the joining instructions on this slide.

Simplified


Processor
Memory
Interconnection network
I/O device 1
I/O device n

slido

Name the I/O devices on the DE1-SoC board

ⓘ Start presenting to display the poll results on this slide.

Accessing I/O Devices

Memory-mapped I/O

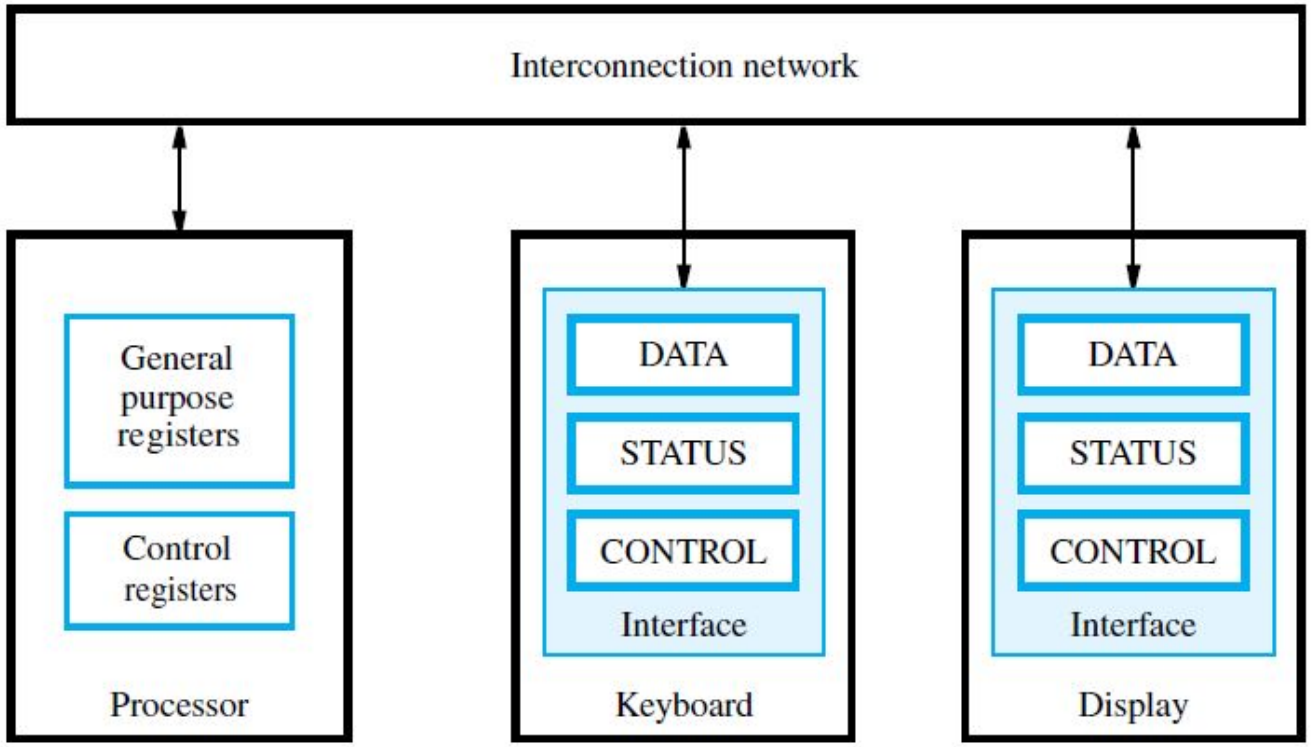
- Each I/O device must appear to the processor as consisting of some addressable locations, just like the memory
- Some addresses in the address space of the processor are assigned to these I/O locations: *I/O registers*.

```
Load R2, DATAIN
Store R2, DATAOUT
```

I/O Device Interface

governs the operation of the device.

- Data registers: a buffer for data transfers
- Status registers: hold info about the current status of the device
- Control registers: hold info that controls the operational behavior of the device



Example: Keyboard

MEF UNIVERSITY
Your Freedom in Learning

slido

Which one can be seen on the DATA register of a keyboard?

ⓘ Start presenting to display the poll results on this slide.

slido

Which one can be seen on the STATUS register of a keyboard?

ⓘ Start presenting to display the poll results on this slide.

Program-Controlled I/O

- Consider a task that reads characters typed on a keyboard, stores these data in the memory, and displays the same characters on a display screen.

Program-Controlled I/O

- Consider a task that reads characters typed on a keyboard, stores these data in the memory, and displays the same characters on a display screen.
- Solution: write a program that performs all functions needed to realize the desired action.
  - transfer each character from the keyboard into the memory
  - transfer each character to the display
  - do it with **right timing**
- **Right timing:**
- The rate of data transfer from the keyboard to a computer is limited by the typing speed of the user: a few characters per second.
- The rate of output transfers from the computer to the display: several thousand characters per sec.
- The difference in speed between the processor and I/O devices creates the need for mechanisms to *synchronize* the transfer of data between them.

Example – Display keyboard character

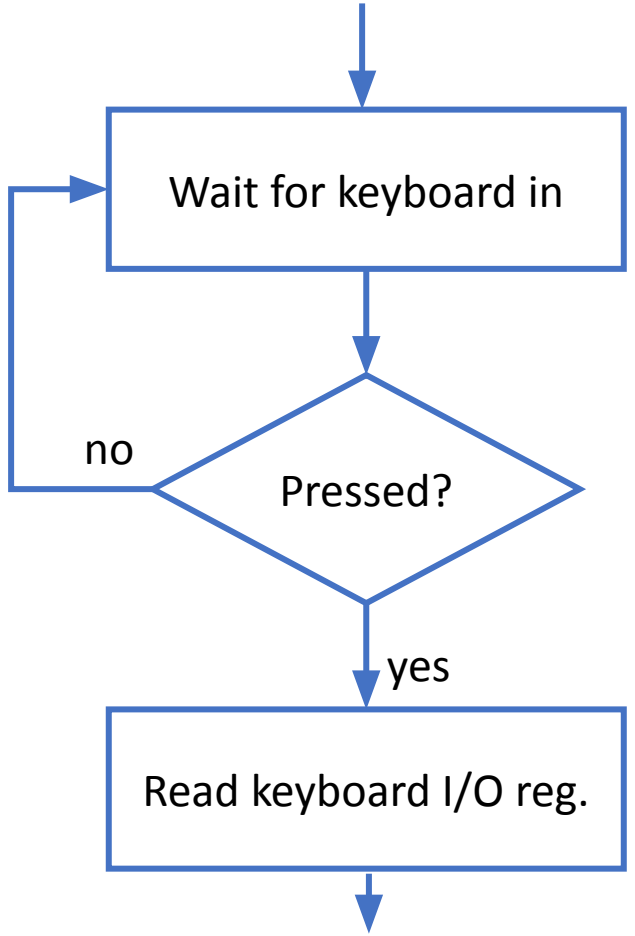
- *ASCII code is used: each character code occupies one byte.*

Example – Display keyboard character

- *ASCII code is used: each character code occupies one byte.*

Keyboard Character Input

- The processor waits for a signal indicating that a key has been pressed.
- A binary code (character) is available in an I/O register associated with the keyboard.
- The processor reads that code.

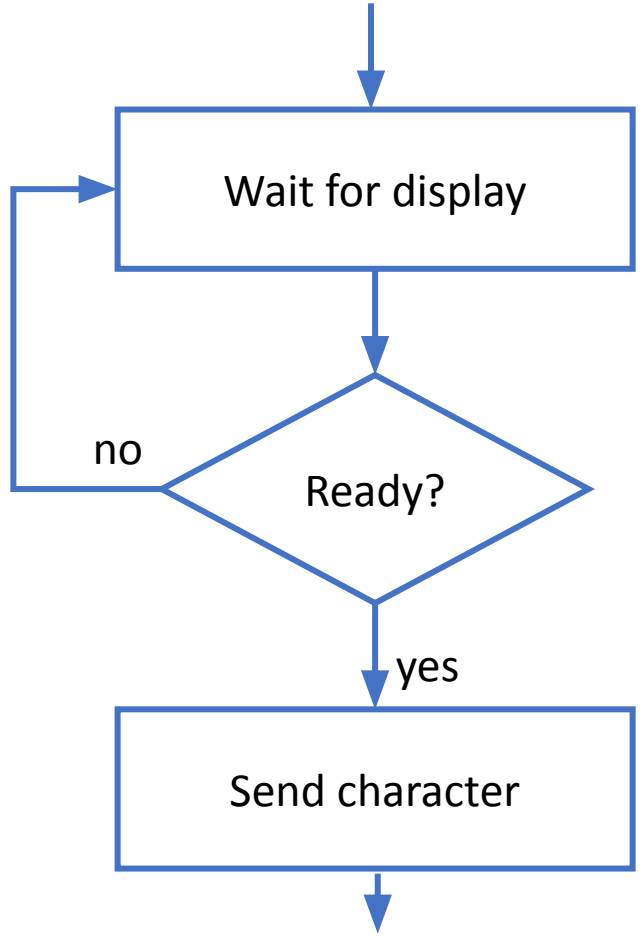


Example – Display keyboard character

Example – Display keyboard character

Display Character Output

- Processor sends the first character
- Waits for a signal from the display that the next character can be sent.
- Then sends the second character.



Example – Display keyboard character

«the processor polls the I/O device»

the processor reads the status flag to determine it state

Keyboard Character Input

status flag **KIN**: a signal indicating that a key has been pressed.

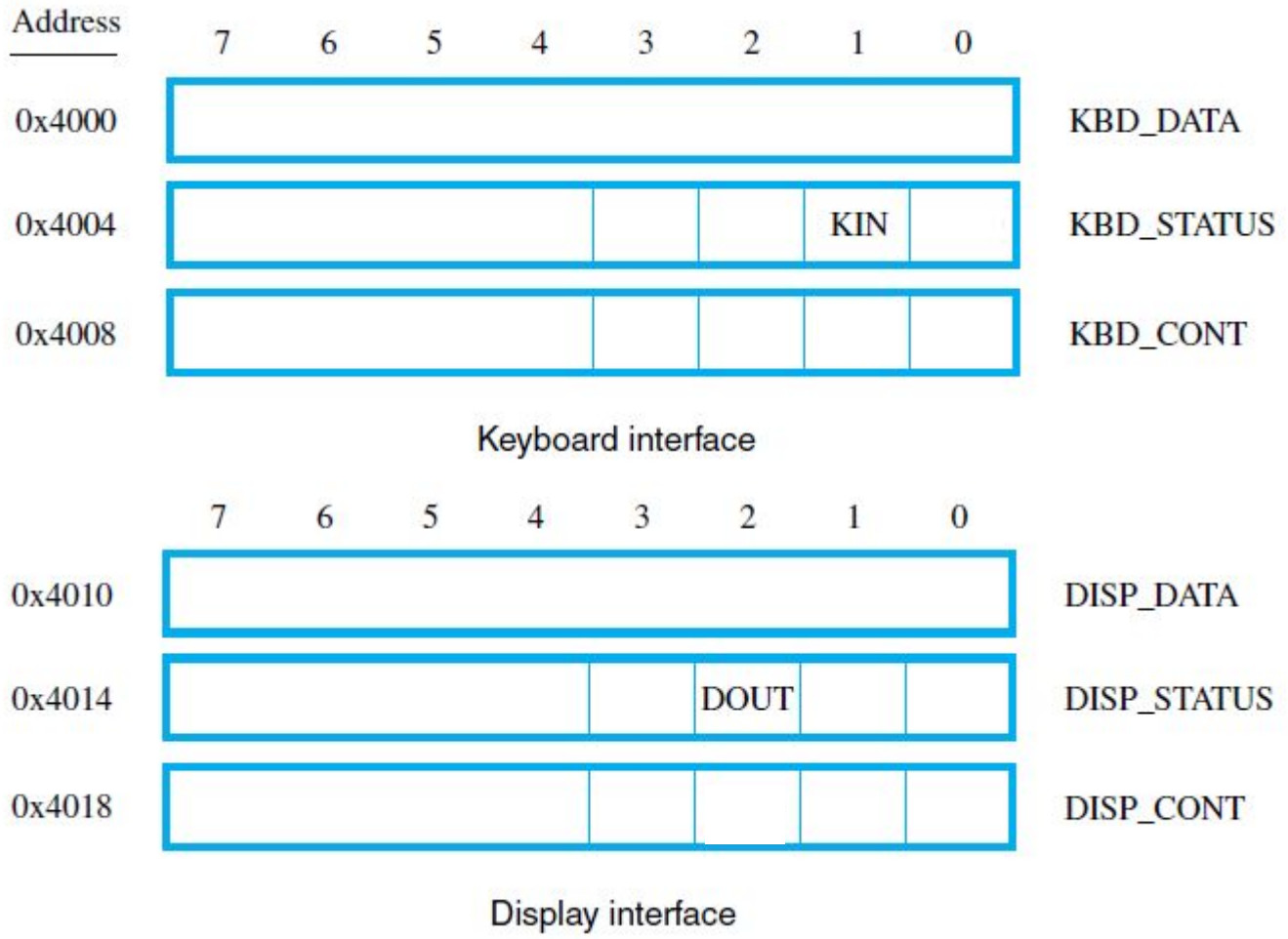
Display Character Output

status flag **DOUT**: it is ready to receive the next character.

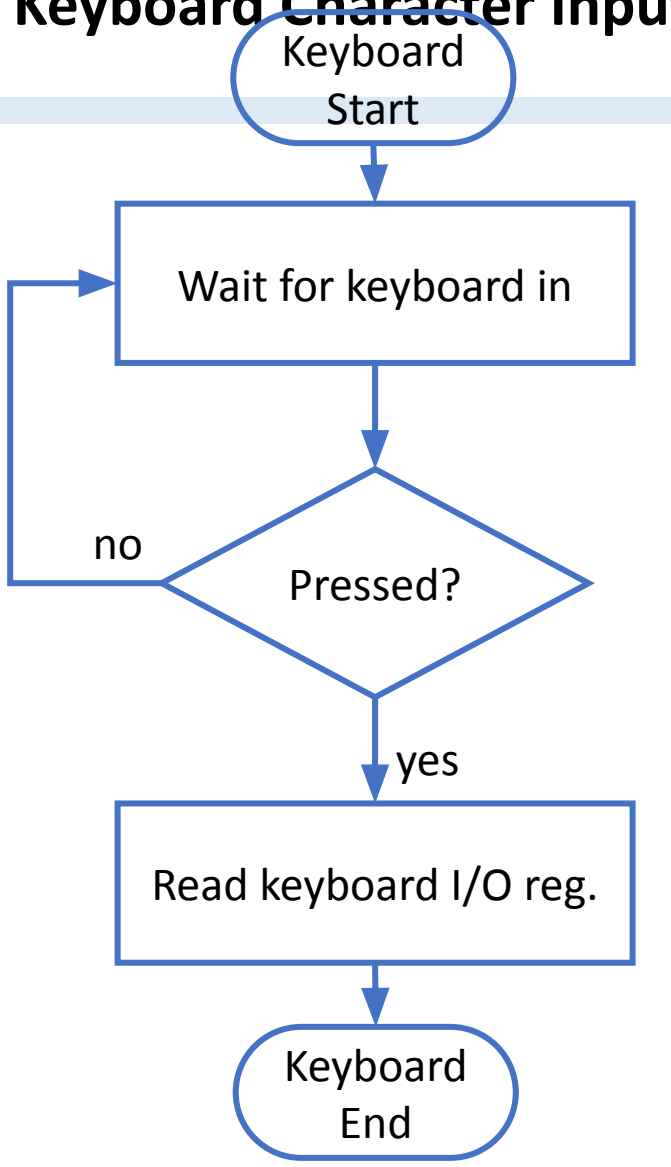
Address

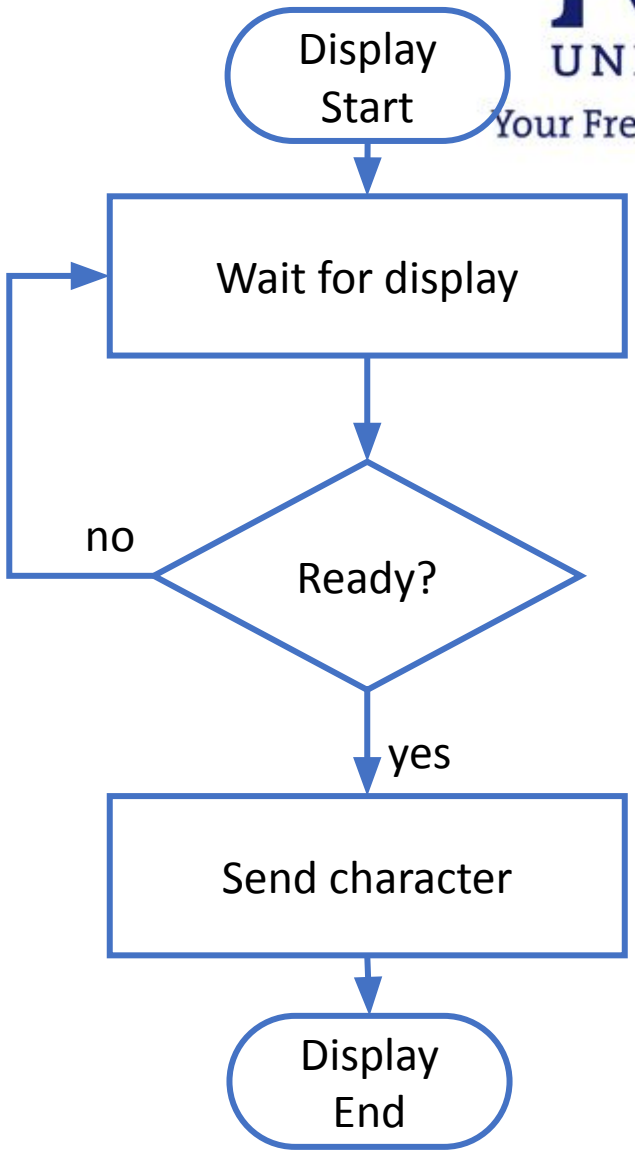
Each register must be assigned a specific address that will be recognized by the interface circuit.

For this example, all addresses word-aligned in a 32-bit word computer.



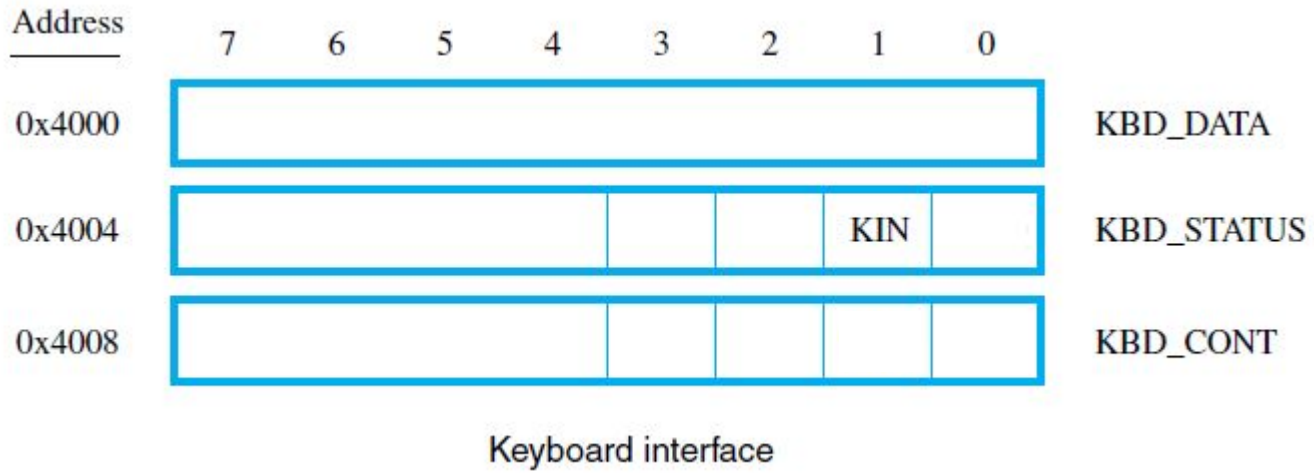
Keyboard Character Input



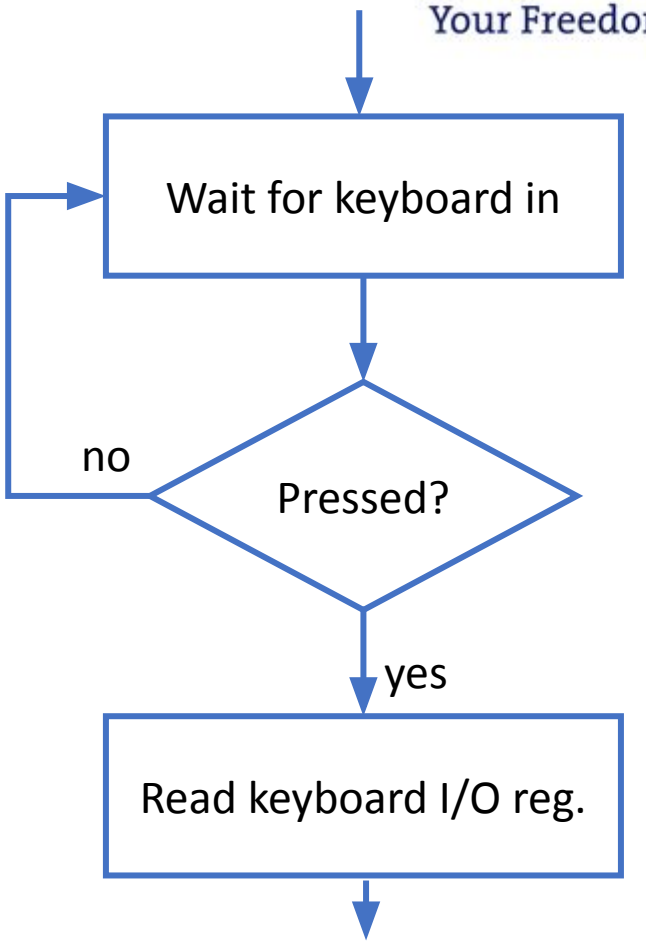


Program-Controlled I/O in ARM

Keyboard Character Input



Keyboard interface



- address KBD_DATA has been loaded into R1
- load the character into R3

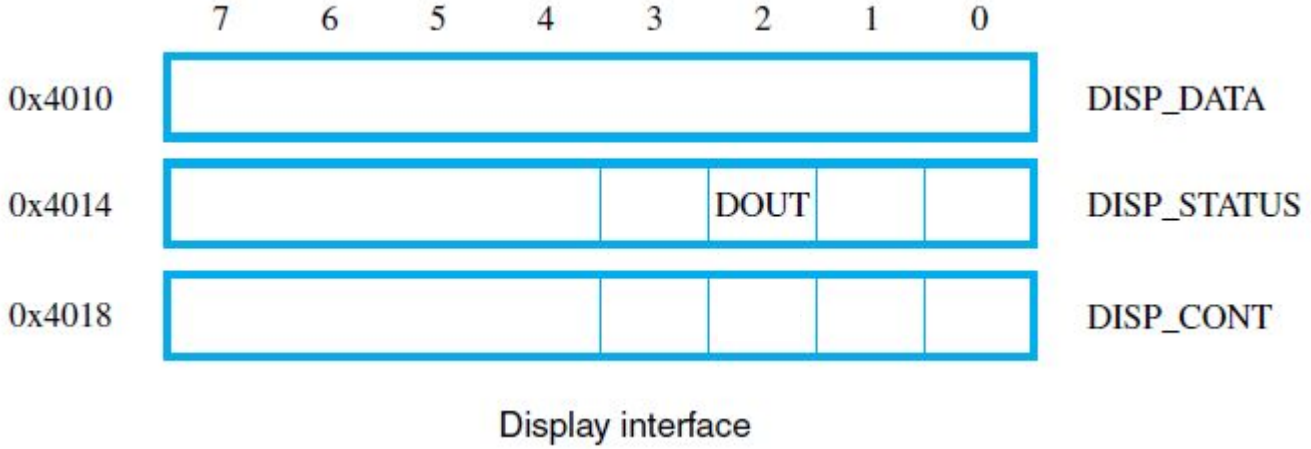
Program-Controlled I/O in ARM

- **Keyboard Character Input**
- address KBD_DATA has been loaded into register R1.

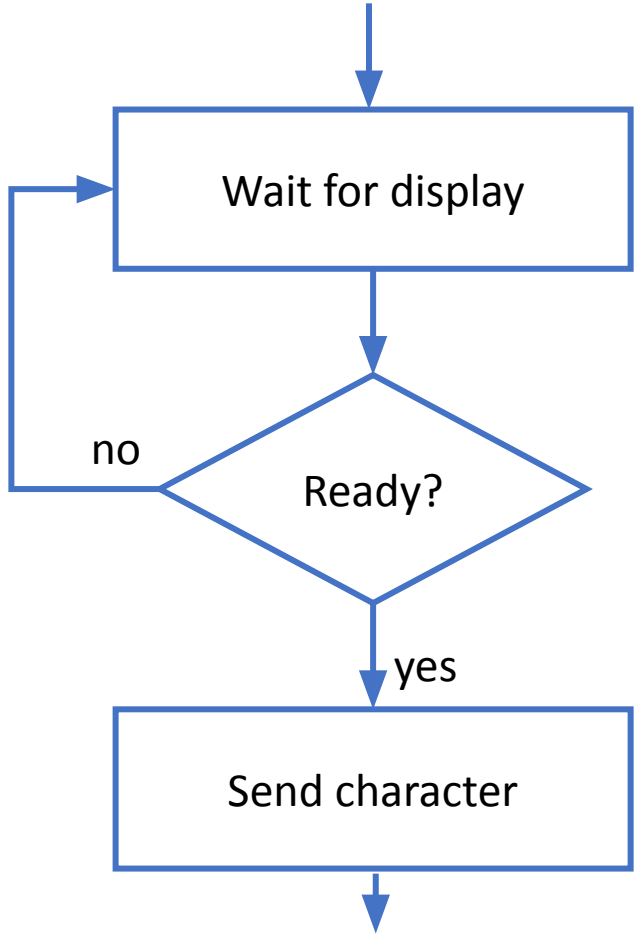
```
READWAIT    LDRB R3, [R1, #4]    // reads a character into register R3
            CMP R3, #2           // be careful! mask if needed.
                                 // the condition code flags are set.
            BNE READWAIT         // back to READWAIT if KIN = 0.
            LDRB R3, [R1]        // else, load the character into R3.
```

Program-Controlled I/O in ARM

Display Character Output



Display interface



- address DISP_DATA has been loaded into register R2

Program-Controlled I/O in ARM

- **Display Character Output**
- address DISP_DATA has been loaded into register R2

```
WRITEWAIT  LDRB R4, [R2, #4]
           CMP R4, #4
           BNE WRITEWAIT
           STRB R3, [R2]
```

//sends the character in register R3 to the DISP_DATA register when the display is ready to receive it.

Program-Controlled I/O in ARM

- Complete Input/Output Program
- R0 contains the address of the first byte in the memory area.

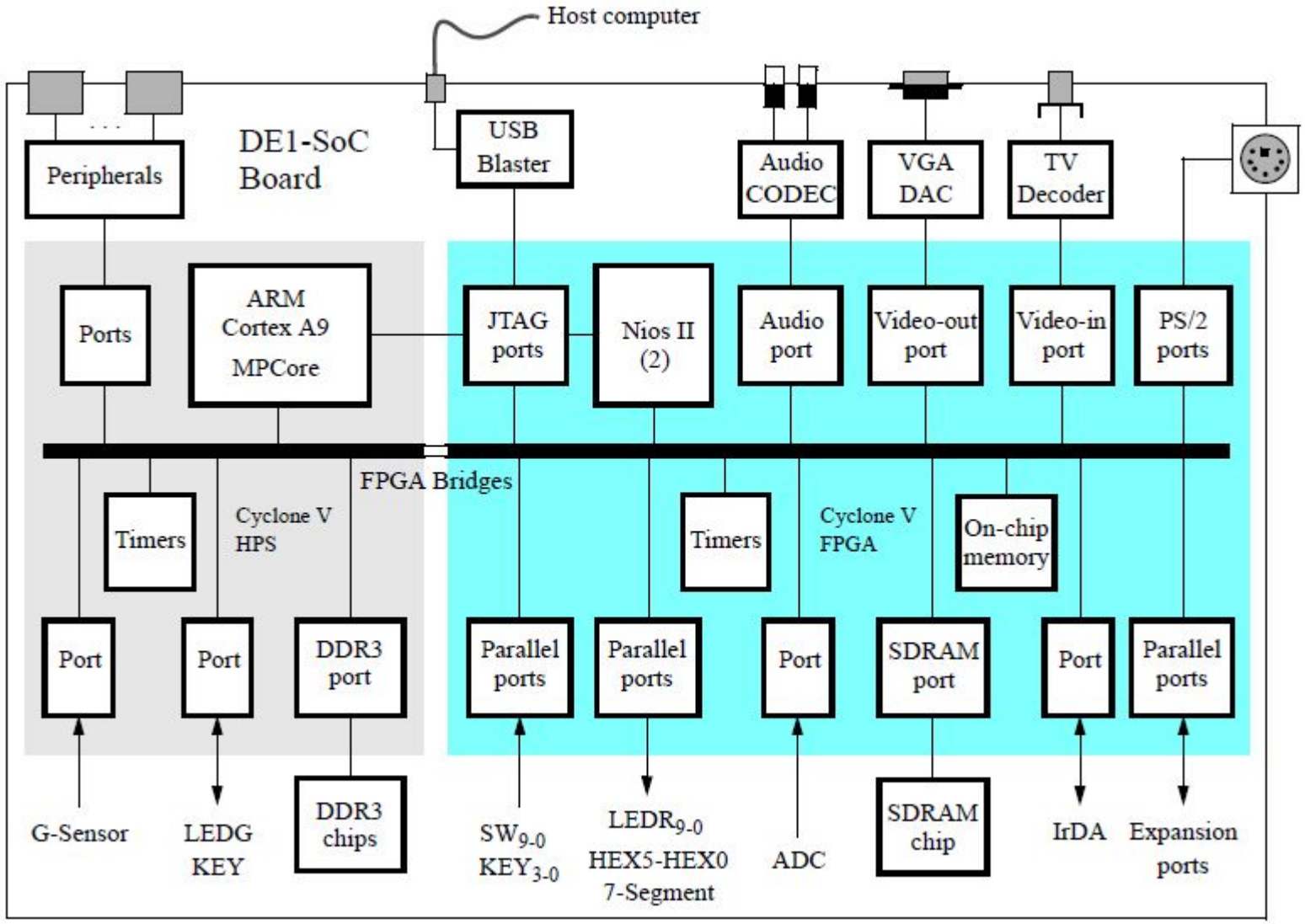
```
READ    LDRB R3, [R1, #4]     //Load KBD_STATUS byte and
        CMP, R3, #2           //    wait for character.
        BNE _______
        LDRB R3, [_____]      // Read the character and
        STRB R3, [R0], #1     //    store it in memory.
ECHO    LDRB R4, [R2, ___]    // Load DISP_STATUS byte and
        CMP R4, #4            //    wait for display
        BNE ____              //    to be ready.
        STRB R3, [____]       // Send character to display.
        ANDS R3, R3, #CR      // If not carriage return,
        BNE ___               //    read more characters.
```

Remarks on program-controlled I/O

- The program enters a wait loop in which it repeatedly tests the device status.
- During this period, the processor is not performing any useful computation.
- We can arrange for the I/O device to alert the processor when it becomes ready.
- Send a hardware signal called an *interrupt request* to the processor
- The processor is no longer required to continuously poll the status of I/O devices waiting periods can ideally be eliminated.

DE1-SoC

See **DE1-SoC Computer System with ARM Cortex-A9 by Altera (Intel)**



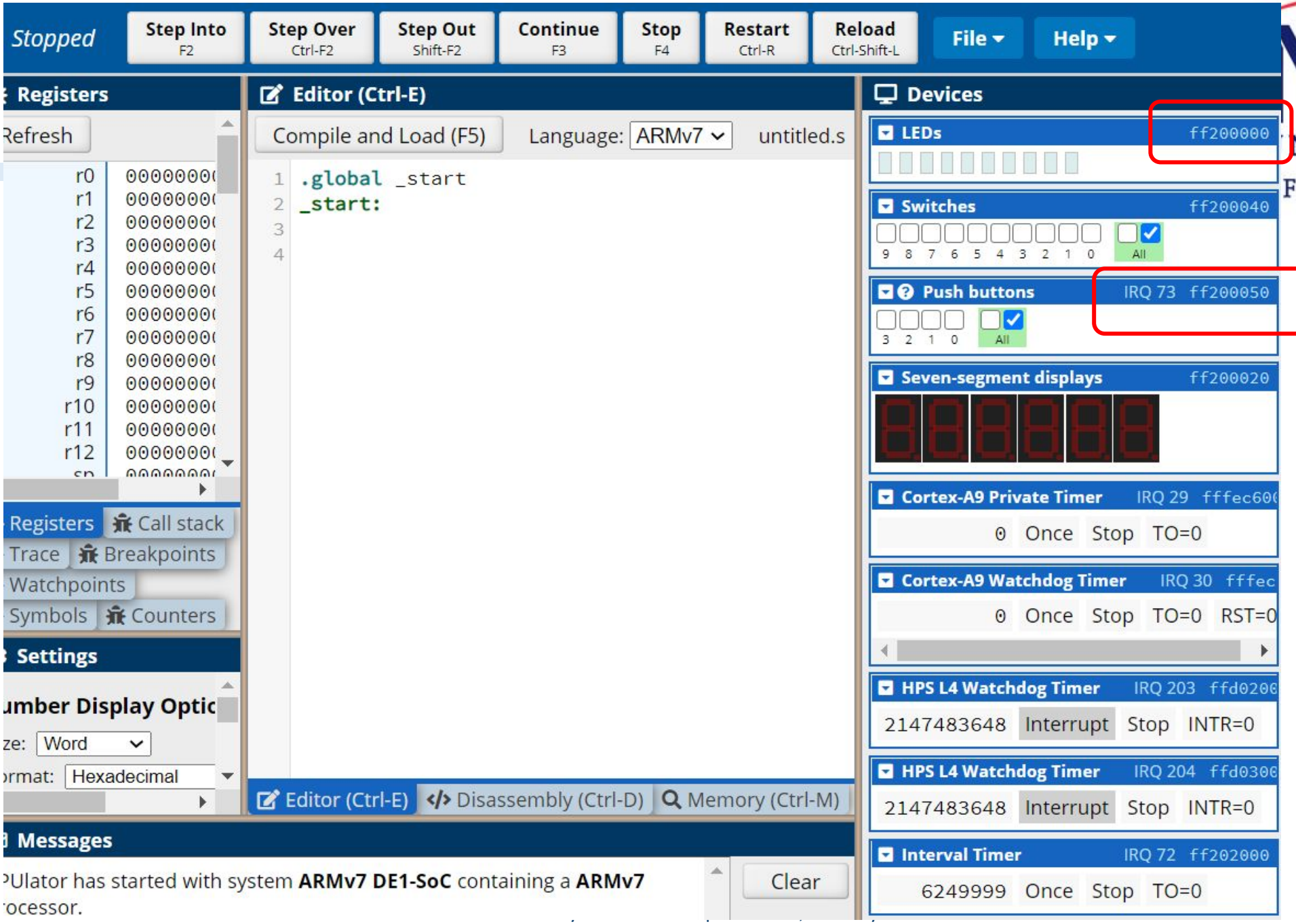
Ref: DE1-SoC Computer System with ARM Cortex-A9

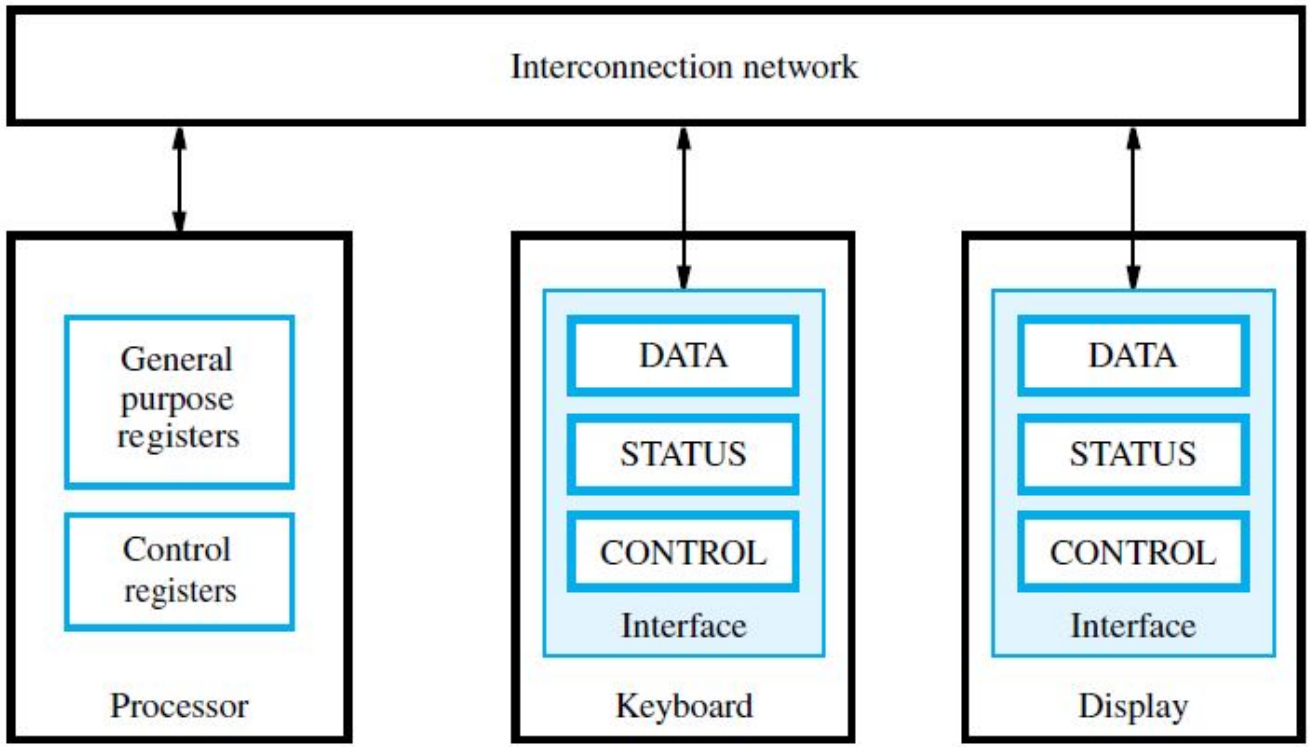
DE1-SoC Computer System

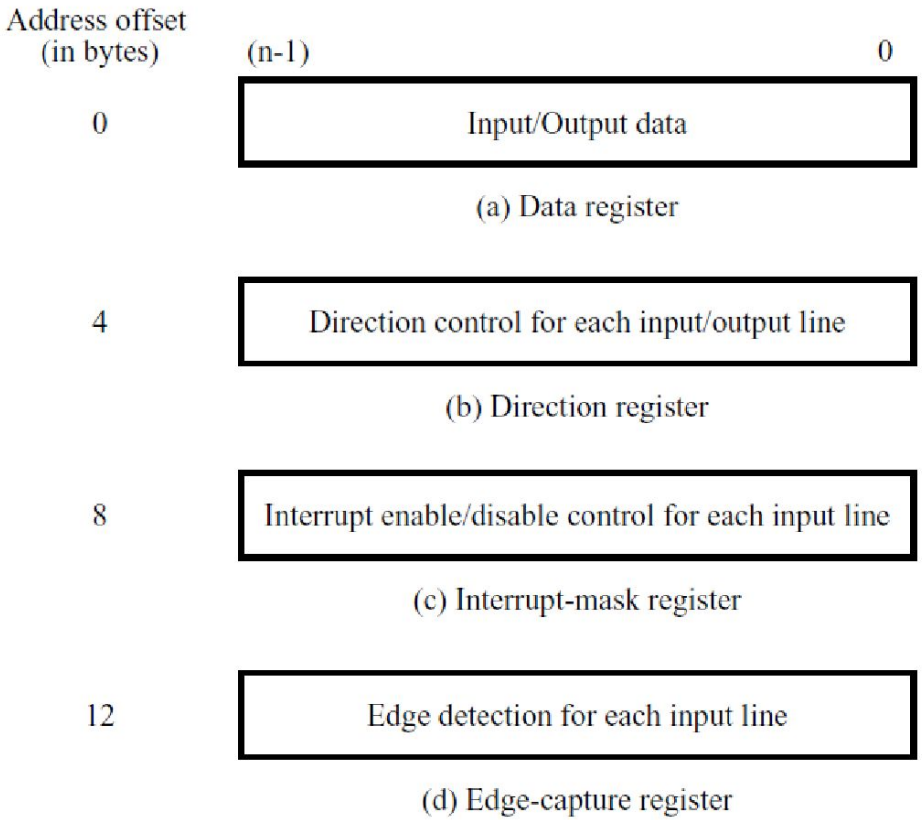
- Hard Processor System (HPS)
  - ARM Cortex A9 dual-core processor
  - DDR3 memory port
  - a set of peripheral devices
- FPGA
  - implements two Altera Nios II processors
  - several peripheral ports: memory, timer modules, audio-in/out, video-in/out, PS/2, analog-to-digital, infrared receive/transmit, and parallel ports connected to switches and lights

Memory

- The memory is organized as 256M x 32-bits
- It is accessible using word accesses (32 bits), halfwords, and bytes
- DDR3 memory is mapped to the address space 0x00000000 to 0x3FFFFFFF.
- 64 KB on-chip memory available inside each ARM A9 processor.
  - Addressed with: FFFF0000 to FFFFFFFF
  - 16K x 32-bits.

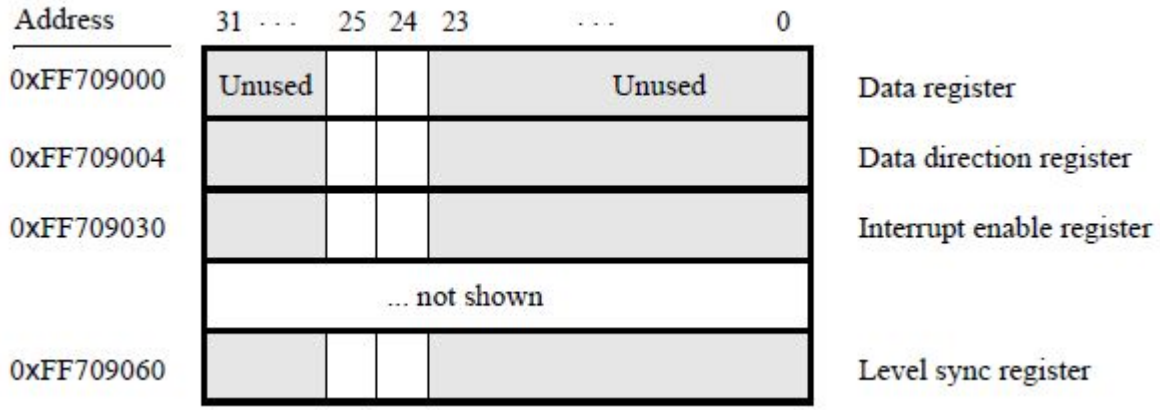



Interconnection network
General purpose registers
Control registers
Processor
DATA
STATUS
CONTROL
Interface
Keyboard
DATA
STATUS
CONTROL
Interface
Display


Address offset (in bytes)
(n-1)
0
0
Input/Output data
(a) Data register
4
Direction control for each input/output line
(b) Direction register
8
Interrupt enable/disable control for each input line
(c) Interrupt-mask register
12
Edge detection for each input line
(d) Edge-capture register

General purpose I/O port: GPIO

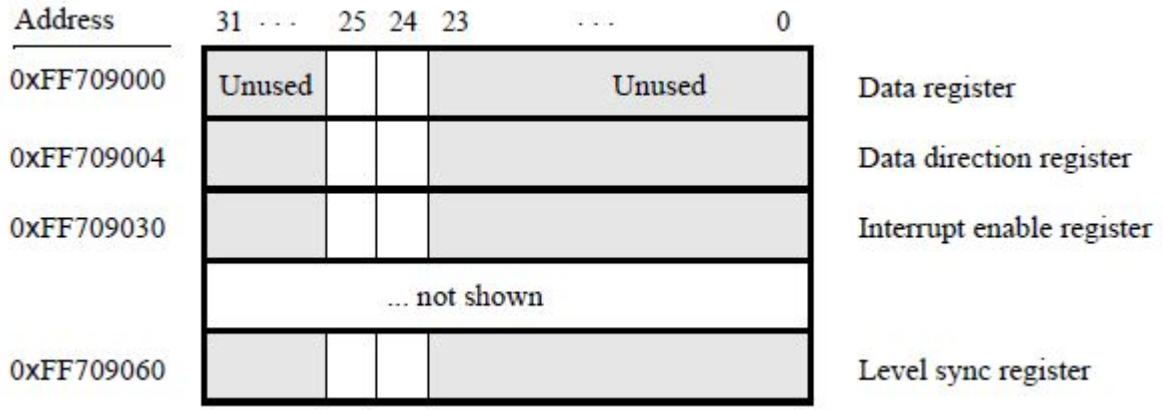
- Pushbutton KEY and LED Port: GPIO1.
- Base address = 0xFF709000
- read or written using word accesses
- Data register (DR)
  - Bit 24: connected to a green light, LEDG
  - Bit 25: connected to a pushbutton switch, KEY
- Data direction register (DDR)
  - configure such that bit 24 is an output and bit 25 is an input:
  - 1: output,
  - 0: input



- LEDG can be turned on/off by writing to bit 24 in the DR.
- The value of the pushbutton switch KEY can be obtained by reading the data register and checking the value of bit 25.

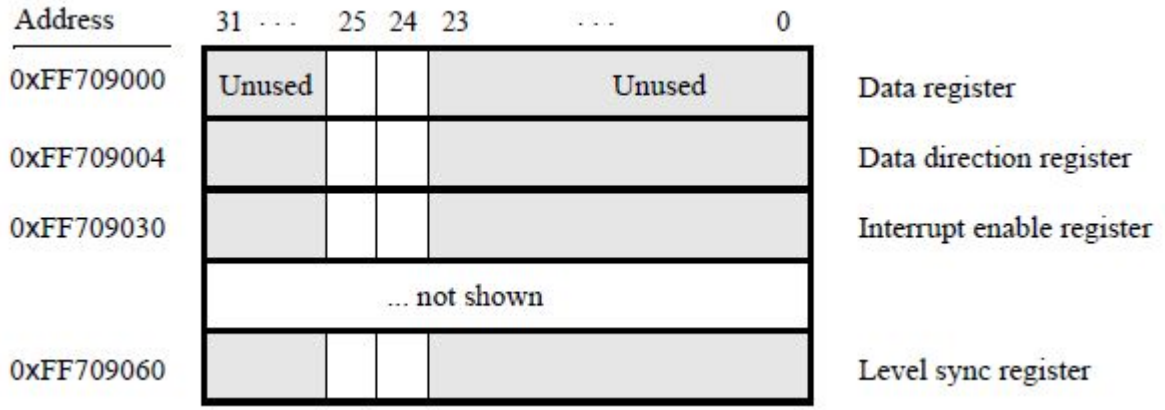
General purpose I/O port: GPIO

- Set the data direction register
  - LEDG (24) is an output → 1
  - KEY (25) is an input → 0



General purpose I/O port: GPIO

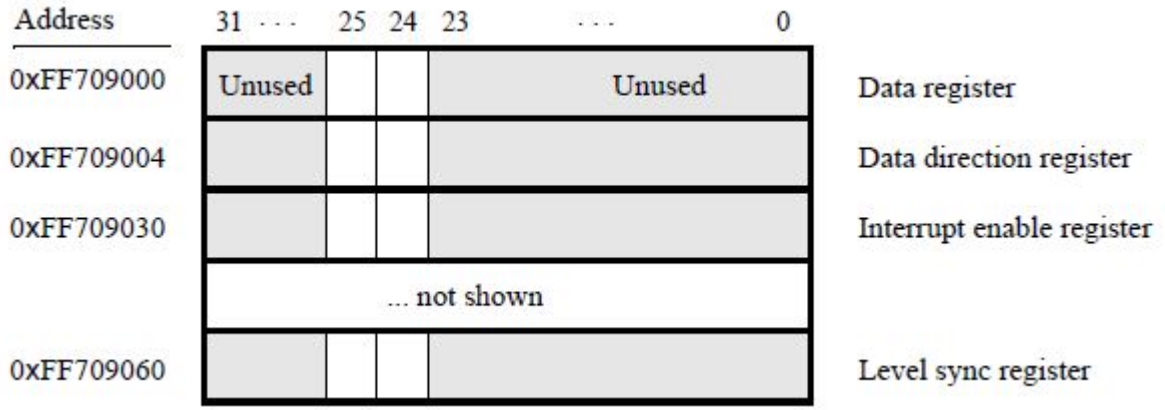
- Set the data direction register
  - LEDG (24) is an output → 1
  - KEY (25) is an input → 0




```
.equ bit_24_pattern, 0x01000000
.global _start
_start:
LDR R0, =0xFF709000      // GPIO1 base address
LDR R2, =bit_24_pattern // value to turn on the HPS green light LEDG
STR R2, [R0, #0x4]       // write to the data direction register to set
                         // bit 24 (LEDG) of GPIO1 to be an output
```

General purpose I/O port: GPIO

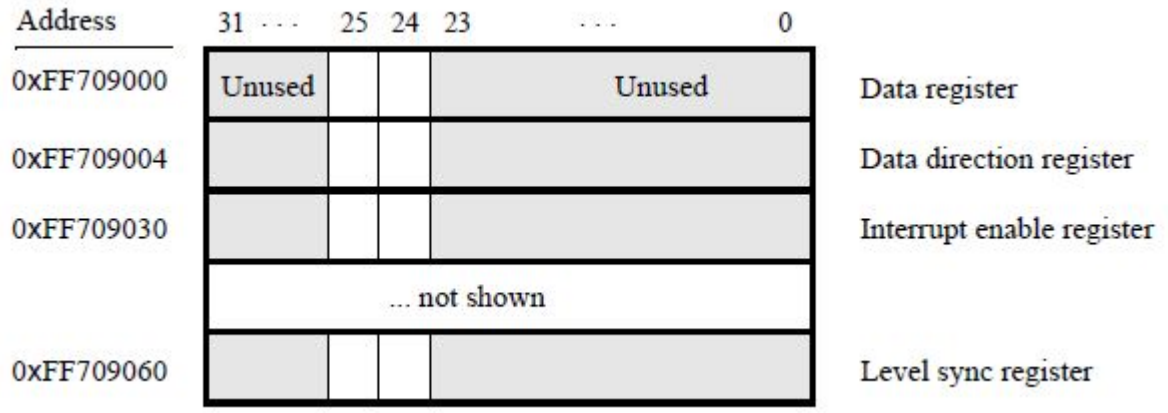
- Turn LEDG on, wait for a while, then turn it off.
- Do this forever.




```
forever:    STR R2, [R0] // turn on/off LEDG
            EOR R2, R2, #bit_24_pattern // toggle LEDG value
            BL Delay_Subroutine
            B forever
```

General purpose I/O port: GPIO

- Turn LEDG on, wait for a while, then turn it off.
- Do this forever, unless the KEY is pressed.




```
forever:    STR R2, [R0] // turn on/off LEDG
            EOR R2, R2, #bit_24_pattern // toggle LEDG value
            BL Delay_Subroutine
            LDR R5, [R0]
            ANDS R5, R5, #bit_25_pattern
            BNE end
            B forever
end: B end
```

slido

Make this program 'better'. Who is the weakest link? Rank from weaker to stronger.

ⓘ Start presenting to display the poll results on this slide.

Timer

fig. e-bay

A timer is a counter.

Imagine a counter counts with a 1 Hz clock.

1 Hz $\longrightarrow$ 1 pulse per second.

If this counter counts for 1 minute, how much will it reach?

Timer

fig. e-bay

A timer is a counter.

Imagine a counter counts with a 1 Hz clock.

1 Hz $\longrightarrow$ 1 pulse per second.

If this counter counts for 1 minute, how much will it reach?

60

Timer

fig. e-bay

A timer is a **DOWN** counter.

Imagine a counter counts with a 1 Hz clock.

1 Hz → 1 pulse per second.

If you want to wait for 60 seconds, which value should you set the clock?

Timer

fig. e-bay

A timer is a **DOWN** counter.

Imagine a counter counts with a 1 Hz clock.

1 Hz → 1 pulse per second.

If you want to wait for 60 seconds, which value should you set the clock?

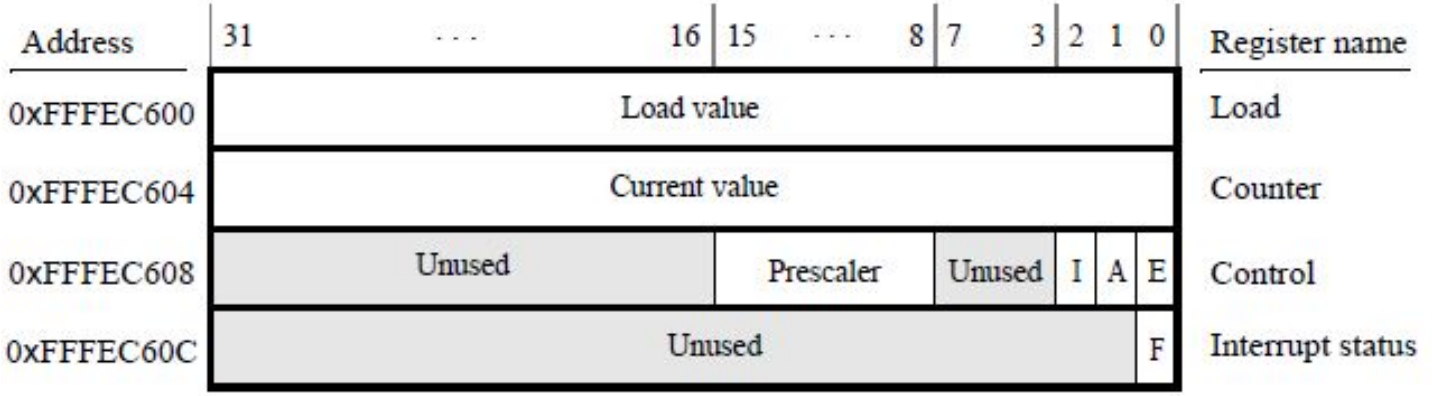
60

Timer Modules

- ARM A9 MPCore includes one private timer module for each A9 core
- HPS provides four other timer modules that can be used by either A9 core.

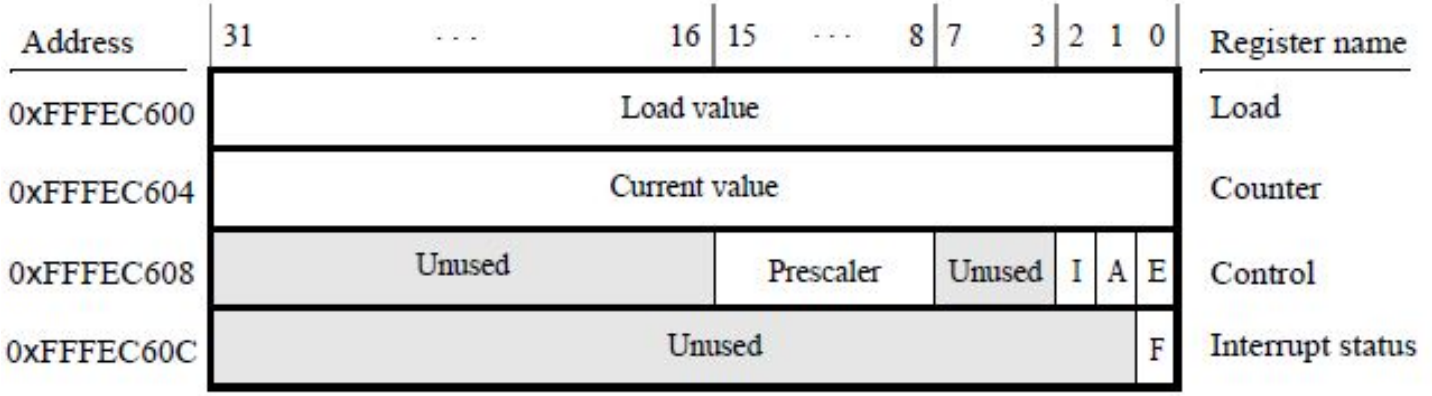
ARM A9 MPCore Timers

- Base address 0xFFFEC600
- It can be read or written using word accesses.
- Clock frequency = 200 MHz



ARM A9 MPCore Timers

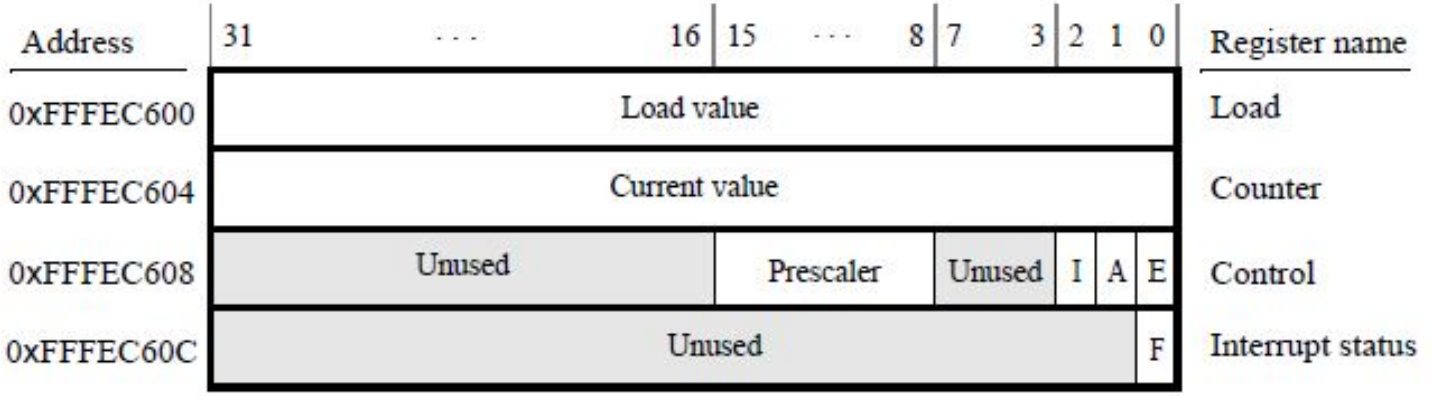
- Base address 0xFFFEC600
- It can be read or written using word accesses.
- Clock frequency = 200 MHz



Set the timer to 200,000,000 (200 Million)
How long does it take to reach 0?

ARM A9 MPCore Timers

- Base address 0xFFFEC600
- It can be read or written using word accesses.
- Clock frequency = 200 MHz



Set the timer to 200,000,000 (200 Million)
How long does it take to reach 0?

1 second.

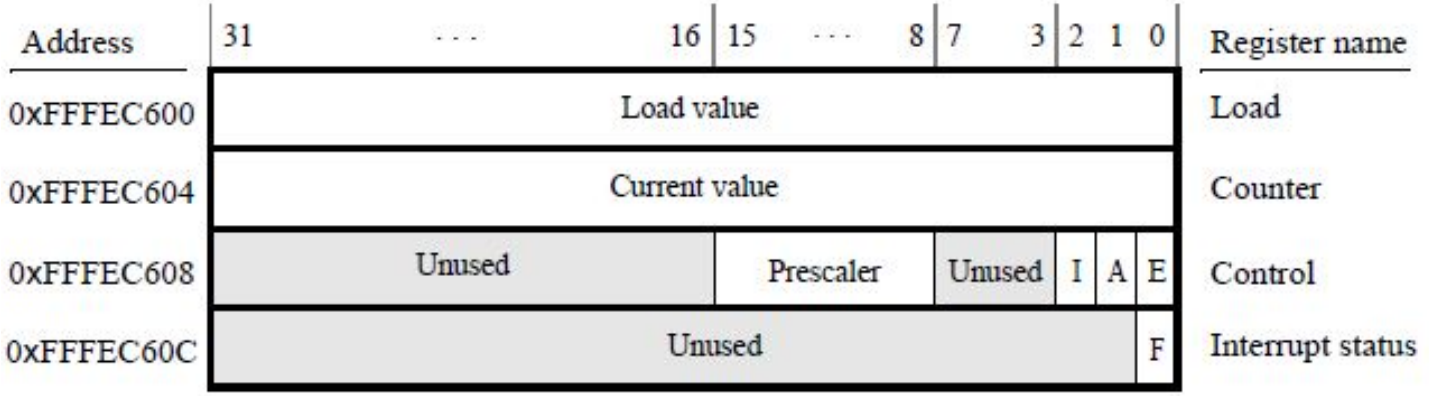
slido

f = 200 MHz. Which value should you set the timer to make a delay of 4 seconds?

ⓘ Start presenting to display the poll results on this slide.

ARM A9 MPCore Timers

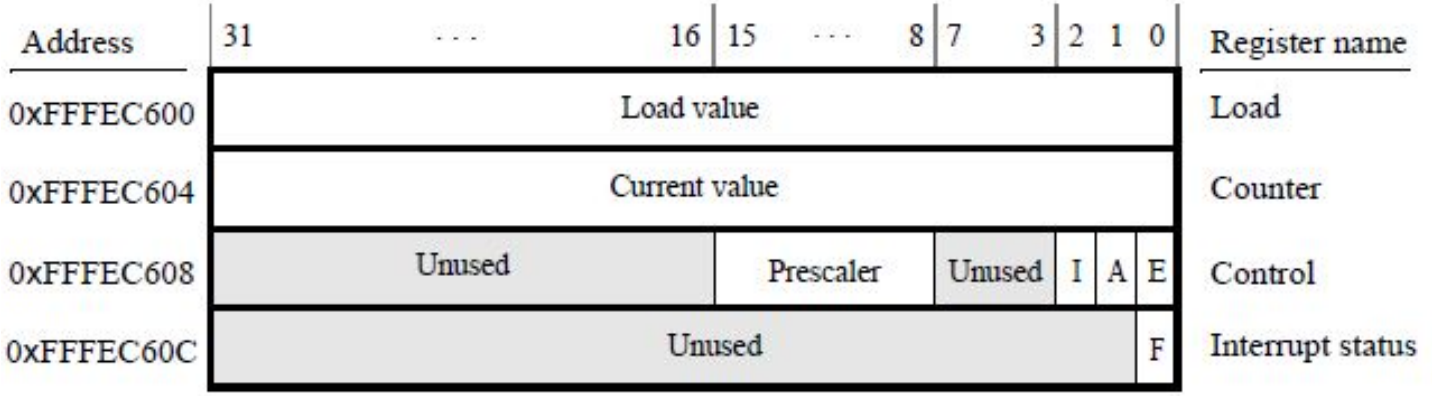
- Base address 0xFFFEC600
- It can be read or written using word accesses.
- Clock frequency = 200 MHz



1. Write an initial count value into the *Load register*

ARM A9 MPCore Timers

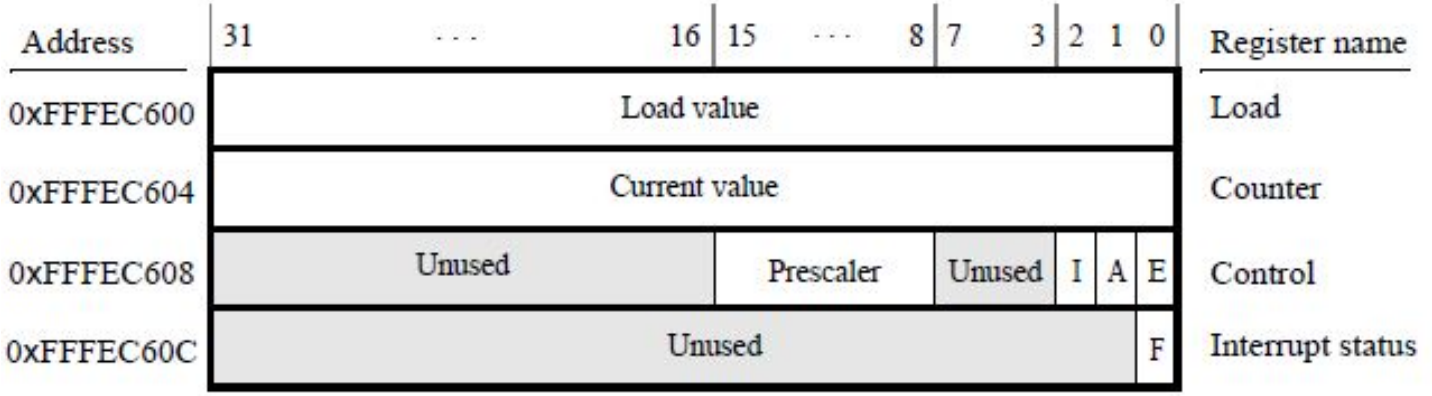
- Base address 0xFFFEC600
- It can be read or written using word accesses.
- Clock frequency = 200 MHz



1. Write an initial count value into the *Load register*
2. Start by setting the enable bit E in the *Control register* to 1.

ARM A9 MPCore Timers

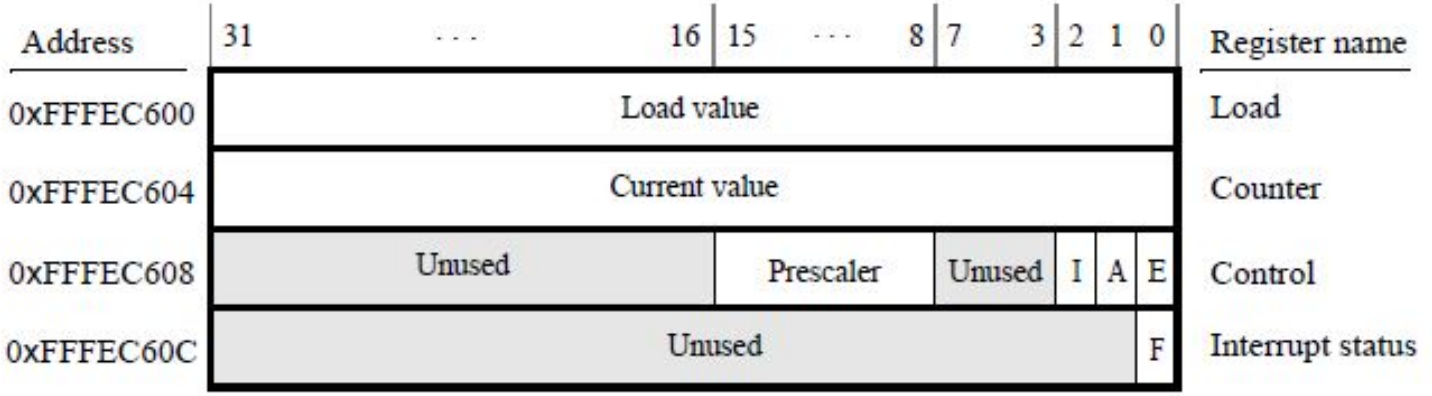
- Base address 0xFFFEC600
- It can be read or written using word accesses.
- Clock frequency = 200 MHz



1. Write an initial count value into the *Load register*
2. Start by setting the enable bit E in the *Control register* to 1.
3. Stop by setting E back to 0.

ARM A9 MPCore Timers

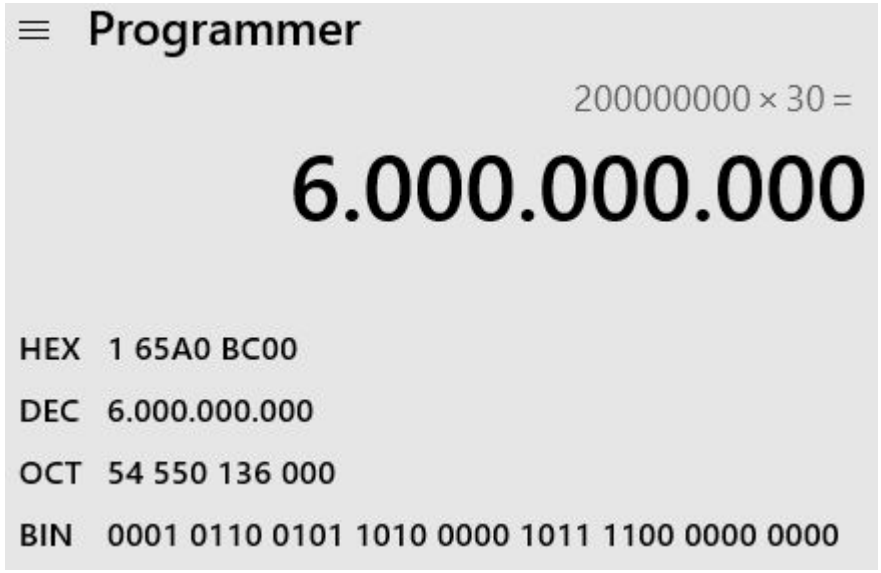
- Timer decrements its count value until reaching 0.
- When it reaches 0, the timer sets the *F* bit The F bit can be checked by software using polled-I/O.
- The F bit can be reset to 0 by writing a 1 into it.
- A bit =1 □ automatically reload the value after the counter reaches 0.



slido

f = 200 MHz. Which value should you set the timer to make a delay of 30 seconds?

ⓘ Start presenting to display the poll results on this slide.

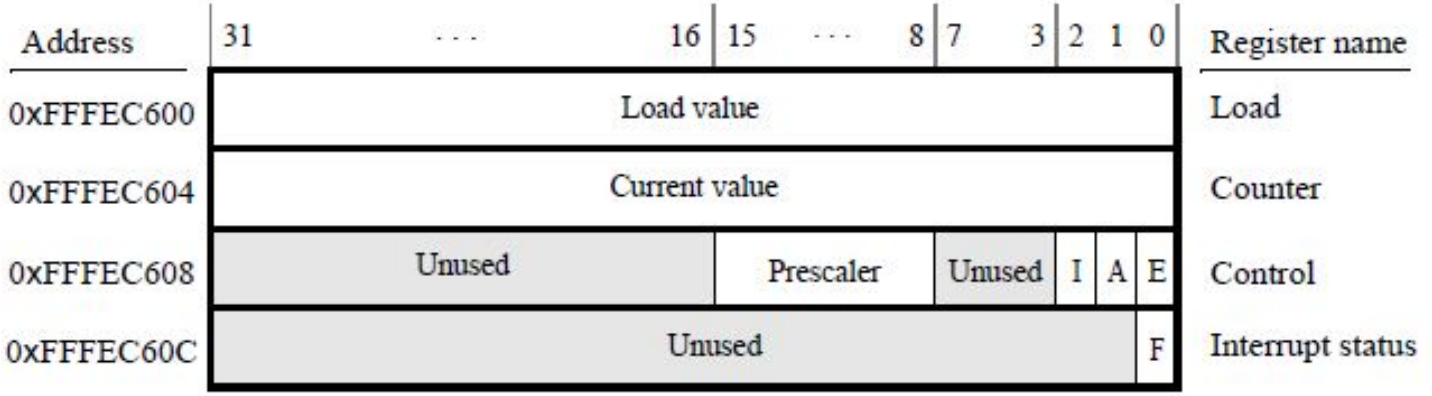


33 bit is required.
However, we have a 32-bit load register.

So, what can we do to generate a 1-minute delay?

ARM A9 MPCore Timers

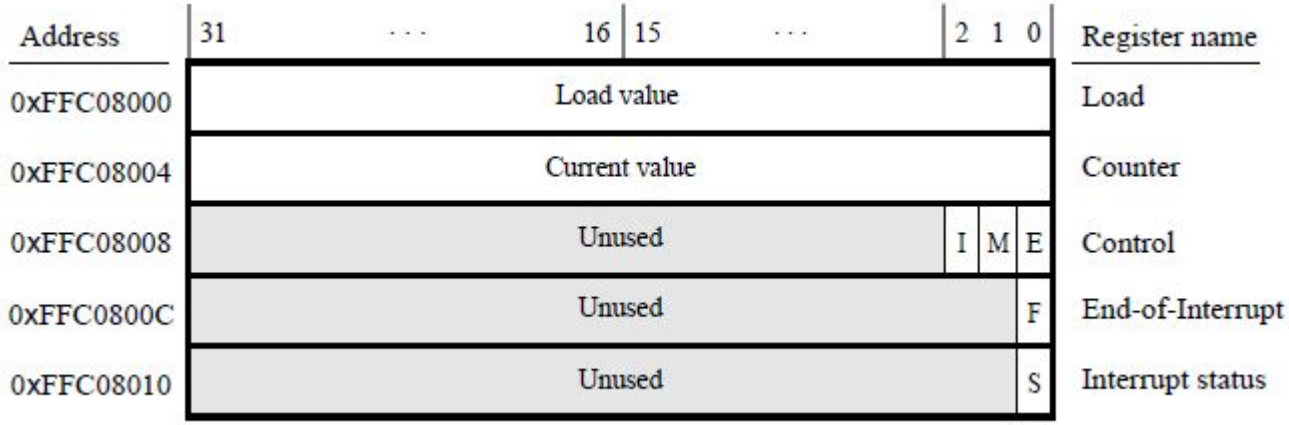
- Timer decrements its count value until reaching 0.
- When it reaches 0, the timer sets the *F* bit The F bit can be checked by software using polled-I/O.
- The F bit can be reset to 0 by writing a 1 into it.
- A bit =1 □ automatically reload the value after the counter reaches 0.



- Prescaler field slows down the counting rate.
  - The timer decrements each Prescaler +1 clock cycle.
  - i.e. Prescaler = 0, then the timer decrements every clock cycle,
  - i.e. Prescaler = 1, the timer decrements every second clock cycle, and so on.

HPS Timers

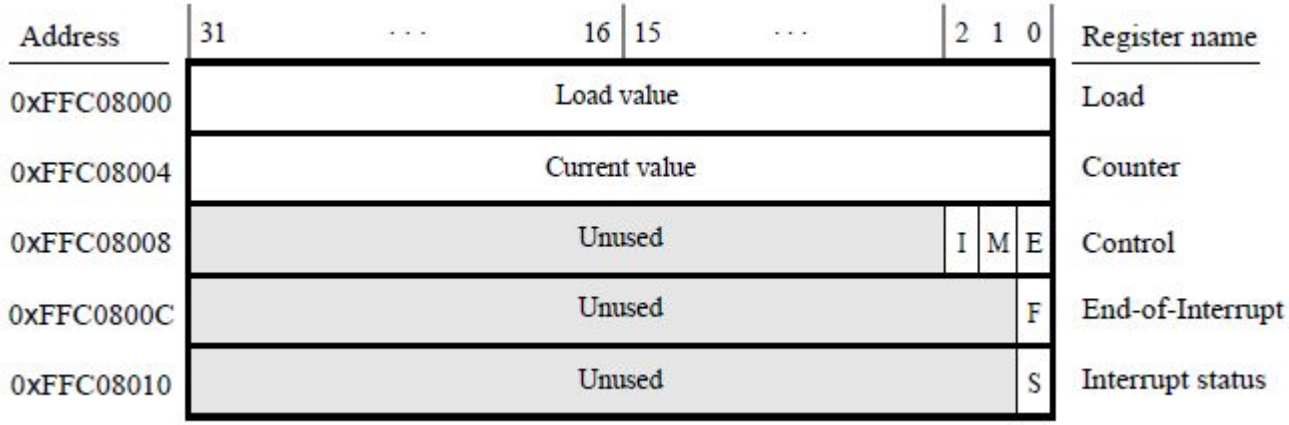
- Read or write using word accesses.
- Base address = 0xFFC08000 □ Clock frequency = 100 MHz
- 0xFFC09000, □ Clock frequency = 100 MHz
- 0xFFD00000 and 0xFFD01000 □ Clock frequency = 25 MHz



1. It is stopped by setting the enable bit E in the Control register to 0.
2. Starting count value for the timer is written into the Load register.

HPS Timers

- Read or write using word accesses.
- Base address = 0xFFC08000 □ Clock frequency = 100 MHz
- 0xFFC09000, □ Clock frequency = 100 MHz
- 0xFFD00000 and 0xFFD01000 □ Clock frequency = 25 MHz

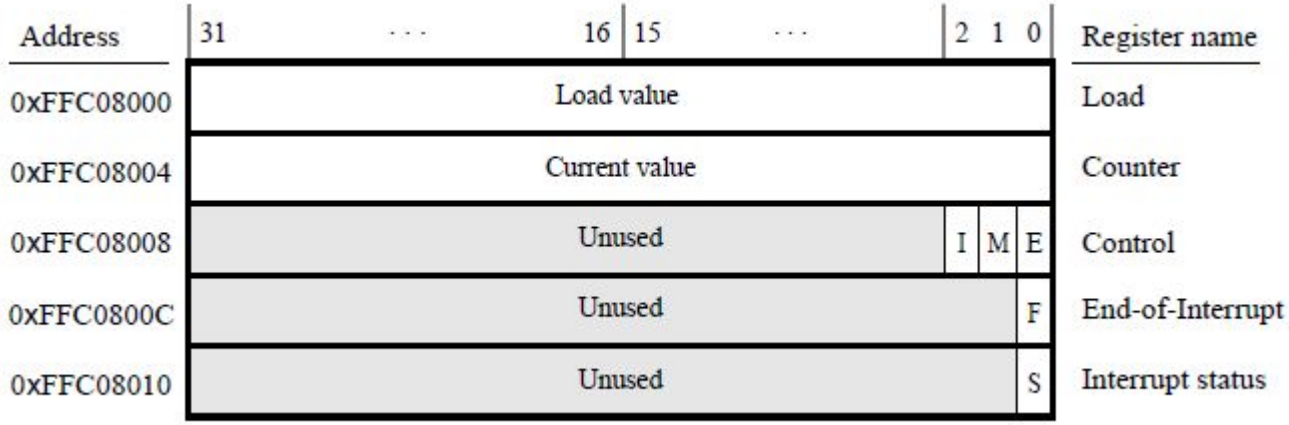


Address	31 ... 16	15 ... 2	1	0	Register name
0xFFC08000	Load value				Load
0xFFC08004	Current value				Counter
0xFFC08008	Unused		I	M E	Control
0xFFC0800C	Unused			F	End-of-Interrupt
0xFFC08010	Unused			S	Interrupt status

1. It is stopped by setting the enable bit E in the Control register to 0.
2. Starting count value for the timer is written into the Load register.
3. Instruct the timer to use the specified starting count value by setting the M in the Control register to 1.
4. The timer can be started by setting E = 1.

HPS Timers

- Read or write using word accesses.
- Base address = 0xFFC08000 □ Clock frequency = 100 MHz
- 0xFFC09000, □ Clock frequency = 100 MHz
- 0xFFD00000 and 0xFFD01000 □ Clock frequency = 25 MHz

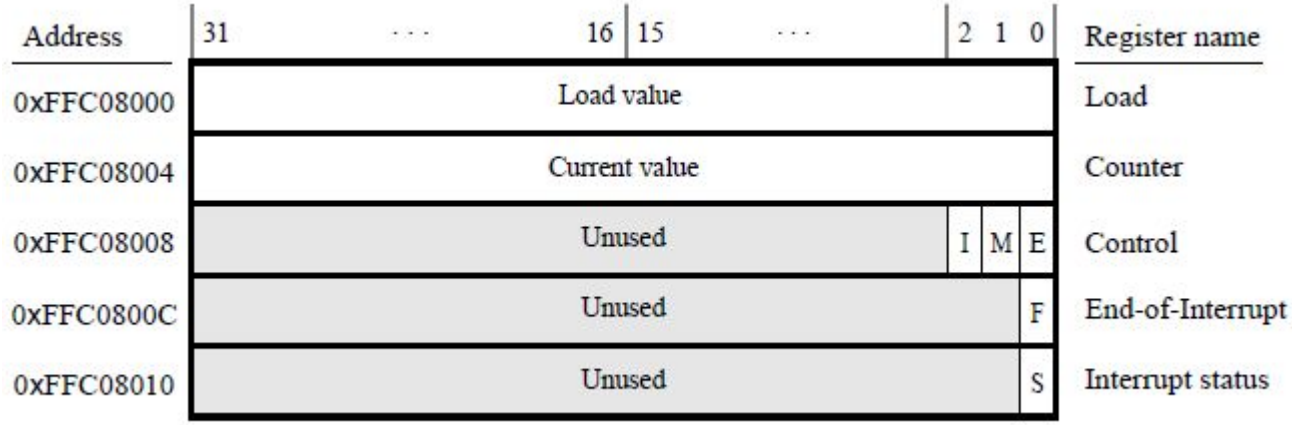


Address	31 ... 16	15 ... 2	1	0	Register name
0xFFC08000	Load value				Load
0xFFC08004	Current value				Counter
0xFFC08008	Unused		I	M E	Control
0xFFC0800C	Unused			F	End-of-Interrupt
0xFFC08010	Unused			S	Interrupt status

1. It is stopped by setting the enable bit E in the Control register to 0.
2. Starting count value for the timer is written into the Load register.
3. Instruct the timer to use the specified starting count value by setting the M in the Control register to 1.
4. The timer can be started by setting E = 1.
5. The timer counts down to 0, and then sets both bit F in the End-of-interrupt register and bit S in the Interrupt status register to 1.

HPS Timers

- Read or write using word accesses.
- Base address = 0xFFC08000 □ Clock frequency = 100 MHz
- 0xFFC09000, □ Clock frequency = 100 MHz
- 0xFFD00000 and 0xFFD01000 □ Clock frequency = 25 MHz

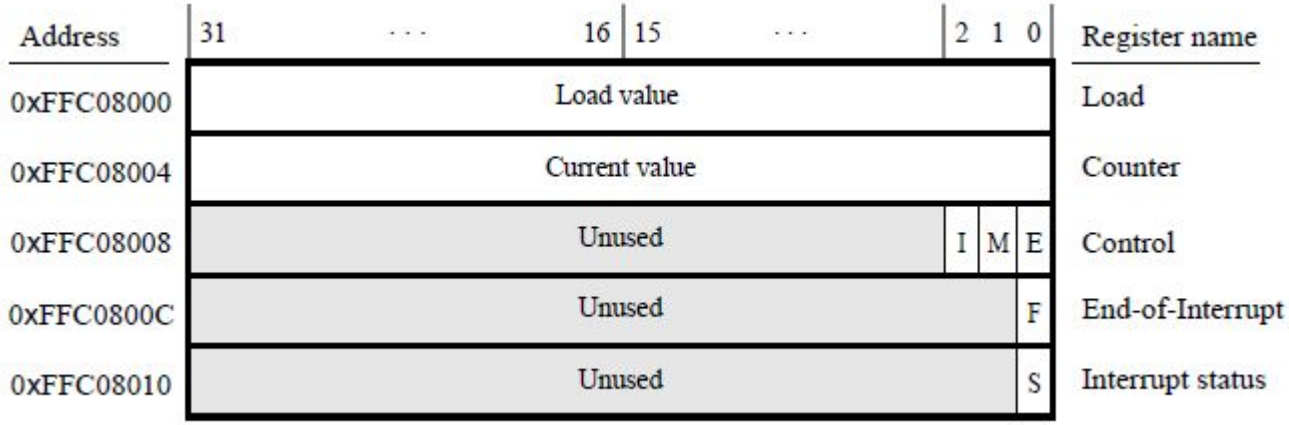


Address	31 ... 16	15 ... 2	1	0	Register name
0xFFC08000	Load value				Load
0xFFC08004	Current value				Counter
0xFFC08008	Unused		I	M	E Control
0xFFC0800C	Unused				F End-of-Interrupt
0xFFC08010	Unused				S Interrupt status

1. It is stopped by setting the enable bit E in the Control register to 0.
2. Starting count value for the timer is written into the Load register.
3. Instruct the timer to use the specified starting count value by setting the M in the Control register to 1.
4. The timer can be started by setting E = 1.
5. The timer counts down to 0, and then sets both bit F in the End-of-interrupt register and bit S in the Interrupt status register to 1.
6. Poll the value of S by software to determine when the timer period has expired.

HPS Timers

- Read or write using word accesses.
- Base address = 0xFFC08000 □ Clock frequency = 100 MHz
- 0xFFC09000, □ Clock frequency = 100 MHz
- 0xFFD00000 and 0xFFD01000 □ Clock frequency = 25 MHz



1. It is stopped by setting the enable bit E in the Control register to 0.
2. Starting count value for the timer is written into the Load register.
3. Instruct the timer to use the specified starting count value by setting the M in the Control register to 1.
4. The timer can be started by setting E = 1.
5. The timer counts down to 0, and then sets both bit F in the End-of-interrupt register and bit S in the Interrupt status register to 1.
6. Poll the value of S by software to determine when the timer period has expired.
7. **!! if bit *I* , the interrupt mask bit, in the Control register is set to 0, then an interrupt can be generated when the timer reaches 0.**

Using a Timer with Assembly Language Code

This program provides a simple example of code for the ARM A9. The program performs the following:

1. starts the ARM A9 private timer
2. loops forever, toggling the HPS green light LEDG when the timer expires

GPIO1

Address	31 ··· 25	24	23 ···	
0xFF709000	Unused		Unused	Data register
0xFF709004				Data direction register
0xFF709030				Interrupt enable register
	... not shown			
0xFF709060				Level sync register

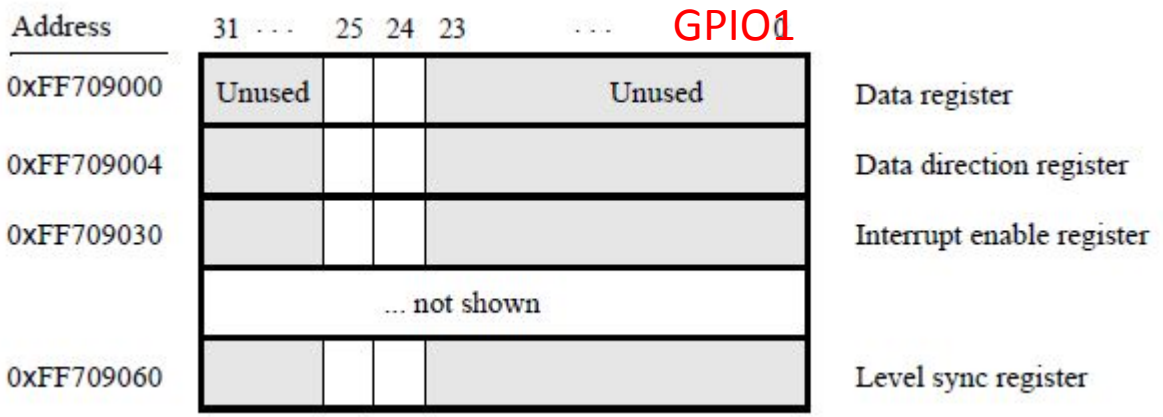
P.Timer

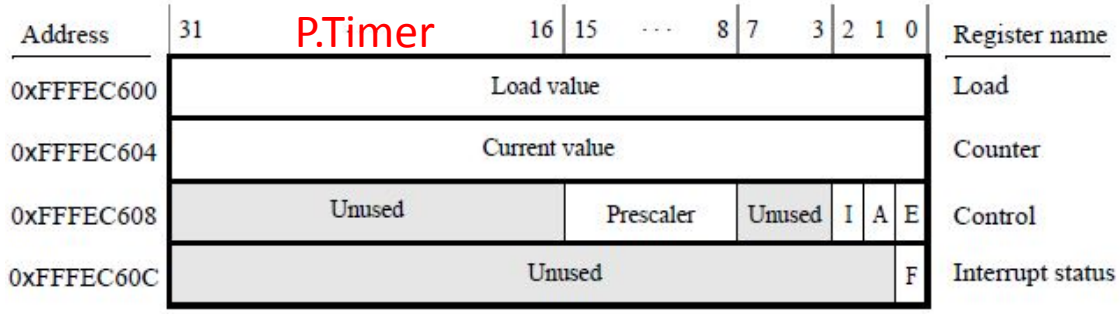
Address	31 ··· 16	15 ··· 8	7 ··· 3	2	1	0	Register name
0xFFFEC600	Load value						Load
0xFFFEC604	Current value						Counter
0xFFFEC608	Unused	Prescaler	Unused	I	A	E	Control
0xFFFEC60C	Unused					F	Interrupt status

Using a Timer with Assembly Language Code

This program provides a simple example of code for the ARM A9. The program performs the following:

1. starts the ARM A9 private timer
2. loops forever, toggling the HPS green light LEDG when the timer expires






```
.equ bit_24_pattern, 0x01000000
.text
.global _start
_start:
LDR R0, =0xFF709000        // GPIO1 base address
LDR R1, =0xFFFEC600        // MPCore private timer base address
LDR R2, =bit_24_pattern    // value to turn on the HPS green light LEDG
STR R2, [R0, #0x4]   // write to the data direction register to set
                     // bit 24 (LEDG) of GPIO1 to be an output
LDR R3, =200000000         // timeout = 1/(200 MHz) x 200x10^6 = 1 sec
STR R3, [R1]             // write to timer load register
MOV R3, #0b011     // set bits: mode = 1 (auto), enable = 1
STR R3, [R1, #0x8]   // write to timer control register
LOOP:
STR R2, [R0]             // turn on/off LEDG
WAIT: LDR R3, [R1, #0xC]   // read timer status
CMP R3, #0
BEQ WAIT                 // wait for timer to expire
STR R3, [R1, #0xC]   // reset timer flag bit
EOR R2, R2, #bit_24_pattern // toggle LEDG value
B LOOP
.end
```

Using a Timer with Assembly Language Code

This program provides a simple example of code for the ARM A9. The program performs the following:

1. starts the ARM A9 private timer
2. loops forever, toggling the HPS green light LEDG when the timer expires

```
.equ bit_24_pattern, 0x01000000
.text
.global _start
_start:
LDR R0, =0xFF709000        // GPIO1 base address
LDR R1, =0xFFFEC600        // MPCore private timer base address
LDR R2, =bit_24_pattern    // value to turn on the HPS green light LEDG
STR R2, [R0, #0x4]   // write to the data direction register to set
                     // bit 24 (LEDG) of GPIO1 to be an output
LDR R3, =200000000         // timeout = 1/(200 MHz) x 200x10^6 = 1 sec
STR R3, [R1]             // write to timer load register
MOV R3, #0b011     // set bits: mode = 1 (auto), enable = 1
STR R3, [R1, #0x8]   // write to timer control register
LOOP:
STR R2, [R0]             // turn on/off LEDG
WAIT: LDR R3, [R1, #0xC]   // read timer status
CMP R3, #0
BEQ WAIT                 // wait for timer to expire
STR R3, [R1, #0xC]   // reset timer flag bit
EOR R2, R2, #bit_24_pattern // toggle LEDG value
B LOOP
.end
```