CEC Continuing Education Center

DesignNews

Getting Started with the Raspberry Pi Pico

DAY 4 : Designing Multicore Raspberry Pi Pico Applications

Sponsored by

Digi-Key ELECTRONICS

BENINGO EMBEDDED GROUP

informa markets

Webinar Logistics

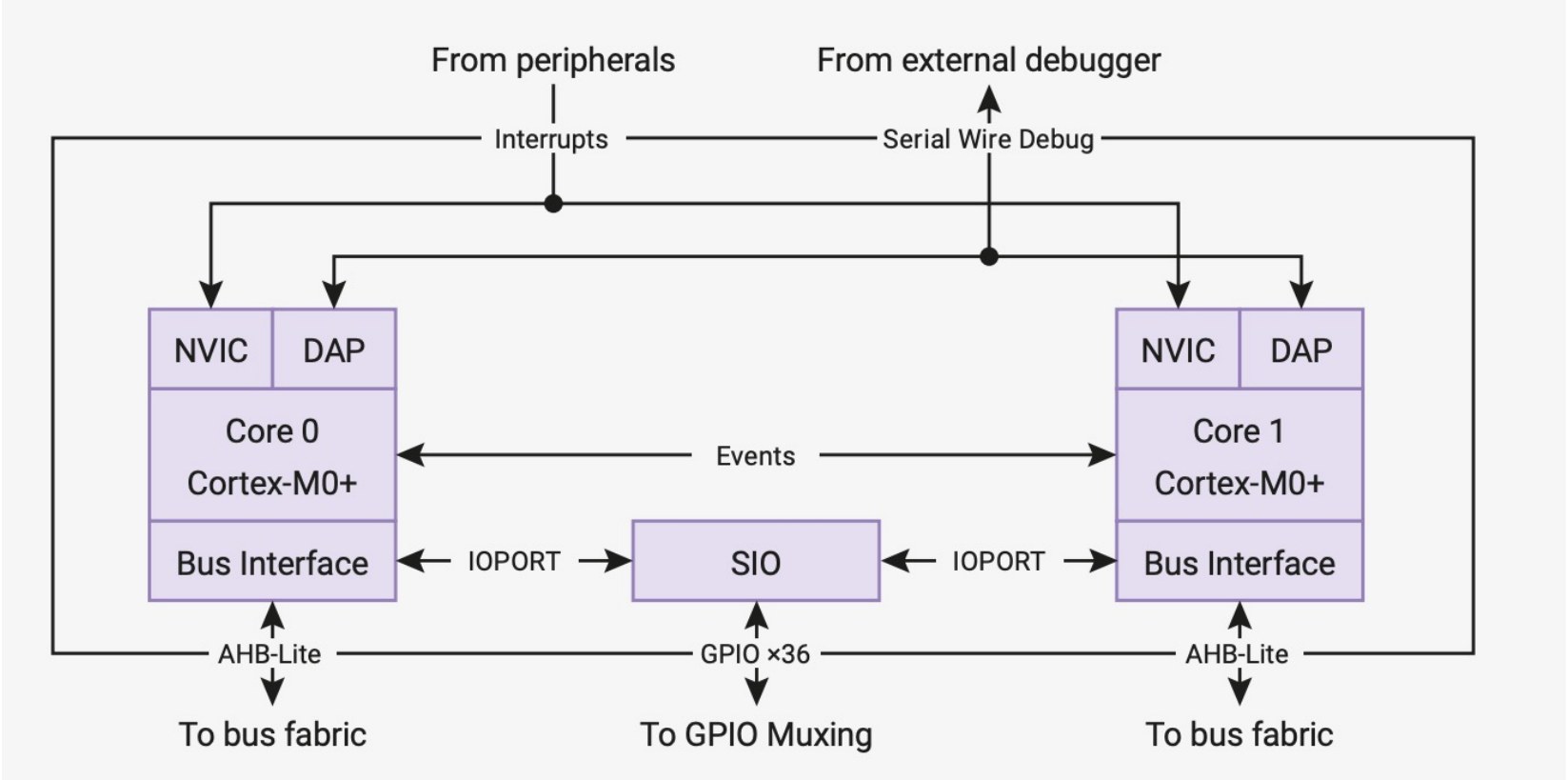
- Turn on your system sound to hear the streaming presentation.
- If you have technical problems, click “Help” or submit a question asking for assistance.
- Participate in ‘Group Chat’ by maximizing the chat widget in your dock.
- Submit questions for the lecturer using the Q&A widget. They will follow-up after the lecture portion concludes.

Course Sessions

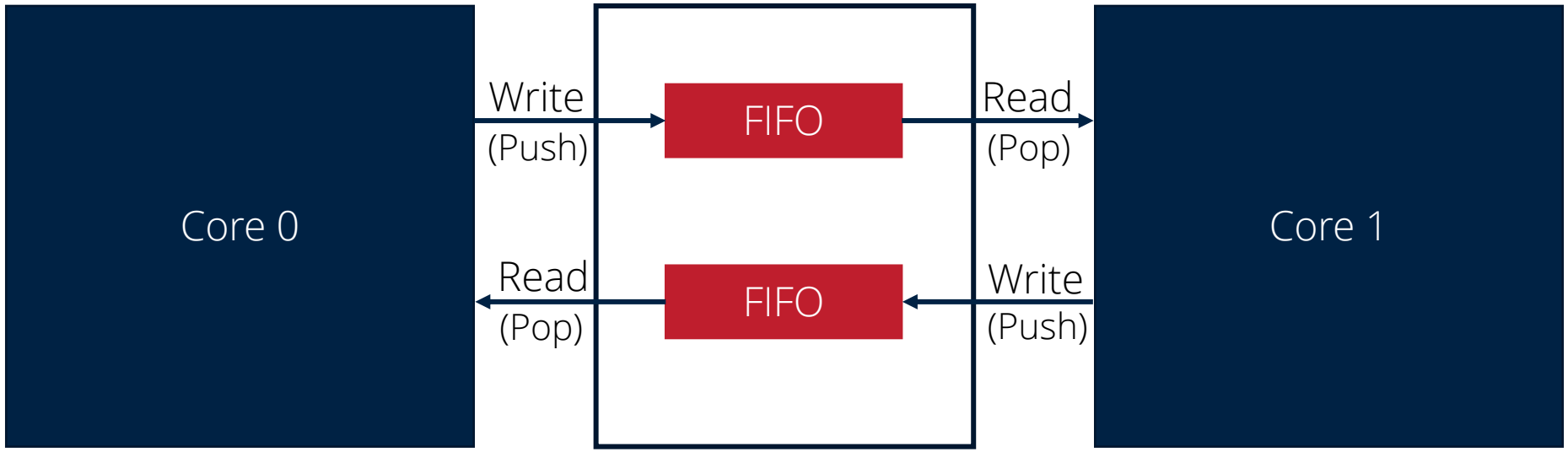
- Introduction to the Raspberry Pi Pico
- Writing your First Raspberry Pi Pico Application
- Interfacing with Raspberry Pi Pico Peripherals
- Designing Multicore Raspberry Pi Pico Applications
- Using MicroPython on the Raspberry Pi Pico

1 The Multicore Architecture

The RP2040 Architecture



Multicore Communication



How experienced are you working with multicore applications?
- No experience
- Beginner
- Competent
- Expert

2 Multicore API's

Modules

- FIFO
  - Provides inter-core functions
- Lockout
  - Function to enable one core to force the other to pause

FIFO

Functions

static bool	multicore_fifo_rvalid (void) Check the read FIFO to see if there is data available (sent by the other core) More...
static bool	multicore_fifo_wready (void) Check the write FIFO to see if it has space for more data. More...
void	multicore_fifo_push_blocking (uint32_t data) Push data on to the write FIFO (data to the other core). More...
bool	multicore_fifo_push_timeout_us (uint32_t data, uint64_t timeout_us) Push data on to the write FIFO (data to the other core) with timeout. More...
uint32_t	multicore_fifo_pop_blocking (void) Pop data from the read FIFO (data from the other core). More...
bool	multicore_fifo_pop_timeout_us (uint64_t timeout_us, uint32_t *out) Pop data from the read FIFO (data from the other core) with timeout. More...
static void	multicore_fifo_drain (void) Discard any data in the read FIFO. More...
static void	multicore_fifo_clear_irq (void) Clear FIFO interrupt. More...
static uint32_t	multicore_fifo_get_status (void) Get FIFO statuses. More...

multicore_fifo_get_status()

```
static uint32_t multicore_fifo_get_status ( void )
```

Get FIFO statuses.

Returns
The statuses as a bitfield

Bit	Description
3	Sticky flag indicating the RX FIFO was read when empty (ROE). This read was ignored by the FIFO.
2	Sticky flag indicating the TX FIFO was written when full (WOF). This write was ignored by the FIFO.
1	Value is 1 if this core's TX FIFO is not full (i.e. if FIFO_WR is ready for more data)
0	Value is 1 if this core's RX FIFO is not empty (i.e. if FIFO_RD is valid)

multicore_fifo_pop_blocking()

```
uint32_t multicore_fifo_pop_blocking ( void )
```

Pop data from the read FIFO (data from the other core).

This function will block until there is data ready to be read Use multicore_fifo_rvalid() to check if data is ready to be read if you don't want to block.

See the note in the fifo section for considerations regarding use of the inter-core FIFOs

Returns
32 bit data from the read FIFO.

multicore_fifo_push_blocking()

```
void multicore_fifo_push_blocking ( uint32_t data )
```

Push data on to the write FIFO (data to the other core).

This function will block until there is space for the data to be sent. Use multicore_fifo_wready() to check if it is possible to write to the FIFO if you don't want to block.

See the note in the fifo section for considerations regarding use of the inter-core FIFOs

Parameters
data A 32 bit value to push on to the FIFO

Lockout

Functions

void	multicore_lockout_victim_init (void) Initialize the current core such that it can be a "victim" of lockout (i.e. forced to pause in a known state by the other core) More...
void	multicore_lockout_start_blocking (void) Request the other core to pause in a known state and wait for it to do so. More...
bool	multicore_lockout_start_timeout_us (uint64_t timeout_us) Request the other core to pause in a known state and wait up to a time limit for it to do so. More...
void	multicore_lockout_end_blocking (void) Release the other core from a locked out state amd wait for it to acknowledge. More...
bool	multicore_lockout_end_timeout_us (uint64_t timeout_us) Release the other core from a locked out state amd wait up to a time limit for it to acknowledge. More...

General Functions

Functions

void	multicore_reset_core1 (void) Reset core 1. More...
void	multicore_launch_core1 (void(*entry)(void)) Run code on core 1. More...
void	multicore_launch_core1_with_stack (void(*entry)(void), uint32_t *stack_bottom, size_t stack_size_bytes) Launch code on core 1 with stack. More...
void	multicore_launch_core1_raw (void(*entry)(void), uint32_t *sp, uint32_t vector_table) Launch code on core 1 with no stack protection. More...

3 Hello Multicore

Hello Multicore

The Goal:

- Write an application that uses both Cortex-M0+ cores
  - Start both cores
  - Transfer data from core 0 to core 1
  - Core 1 receive core 0 data and “process it”
  - Print results on stdio

Hello Multicore!

```
#include <stdio.h>
#include "pico/stdlib.h"
#include "pico/multicore.h”

#define FLAG_VALUE 123

void core1_entry() {

    multicore_fifo_push_blocking(FLAG_VALUE);

    uint32_t g = multicore_fifo_pop_blocking();

    if (g != FLAG_VALUE)
        printf("Hmm, that's not right on core 1!\n");
    else
        printf("Its all gone well on core 1!");

    while (1)
        tight_loop_contents();
}
```

```
int main() {
    stdio_init_all();
    printf("Hello, multicore!\n");

    multicore_launch_core1(core1_entry);

    // Wait for it to start up

    uint32_t g = multicore_fifo_pop_blocking();

    if (g != FLAG_VALUE)
        printf("Hmm, that's not right on core 0!\n");
    else {
        multicore_fifo_push_blocking(FLAG_VALUE);
        printf("It's all gone well on core 0!");
    }
}
```

What function is used to initialize core 0?
- multicore_launch_core0
- multicore_launch_core1
- main
- other

3 More Multicore Example(s)

Multicore FIFO with SIO

The Goal:

- Write an application that uses both Cortex-M0+ cores
  - Start both cores
  - Transfer data from core 0 to core 1
  - Use the SIO Interrupt

Core Code

```
#include <stdio.h>
#include "pico/stdlib.h"
#include "pico/multicore.h"
#include "hardware/irq.h”

#define FLAG_VALUE 123

void core1_entry() {
    multicore_fifo_clear_irq();
    irq_set_exclusive_handler(SIO_IRQ_PROC1, core1_sio_irq);

    irq_set_enabled(SIO_IRQ_PROC1, true);

    // Send something to Core0, this should fire the interrupt.
    multicore_fifo_push_blocking(FLAG_VALUE1);

    while (1)
        tight_loop_contents();
}
```

```
int main() {
    stdio_init_all();
    printf("Hello, multicore_fifo_irqs!\n");

    // We MUST start the other core before we enabled FIFO interrupts.
    // This is because the launch uses the FIFO's, enabling interrupts before
    // they are used for the launch will result in unexpected behaviour.
    multicore_launch_core1(core1_entry);

    irq_set_exclusive_handler(SIO_IRQ_PROC0, core0_sio_irq);
    irq_set_enabled(SIO_IRQ_PROC0, true);

    // Wait for a bit for things to happen
    sleep_ms(10);

    // Send something back to the other core
    multicore_fifo_push_blocking(FLAG_VALUE2);

    // Wait for a bit for things to happen
    sleep_ms(10);

    printf("Irq handlers should have rx'd some stuff - core 0 got %d, core 1 got %d!\n", core0_rx_val, core1_rx_val);

    while (1)
        tight_loop_contents();
}
```

SIO Interrupts

```
void core0_sio_irq()
{
    // Just record the latest entry
    while (multicore_fifo_rvalid())
    {
        core0_rx_val = multicore_fifo_pop_blocking();
    }

    multicore_fifo_clear_irq();
}
```

```
 void core1_sio_irq()
{
    // Just record the latest entry
    while (multicore_fifo_rvalid())
    {
        core1_rx_val = multicore_fifo_pop_blocking();
    }

    multicore_fifo_clear_irq();
}
```

More Multicore Examples

Are you going to run the multicore examples on the Pico?
- Yes
- No
- Not sure

4 Going Further

Thank you for attending

Please consider the resources below:

- www.beningo.com
  - Blog, White Papers, Courses
  - Embedded Bytes Newsletter
    - http://bit.ly/1BAHYXm

From www.beningo.com under

- Blog > CEC – Getting Started with the Raspberry Pi Pico

CEC Continuing Education Center

DesignNews

Thank You

Sponsored by

Digi-Key ELECTRONICS

BENINGO EMBEDDED GROUP

informa markets