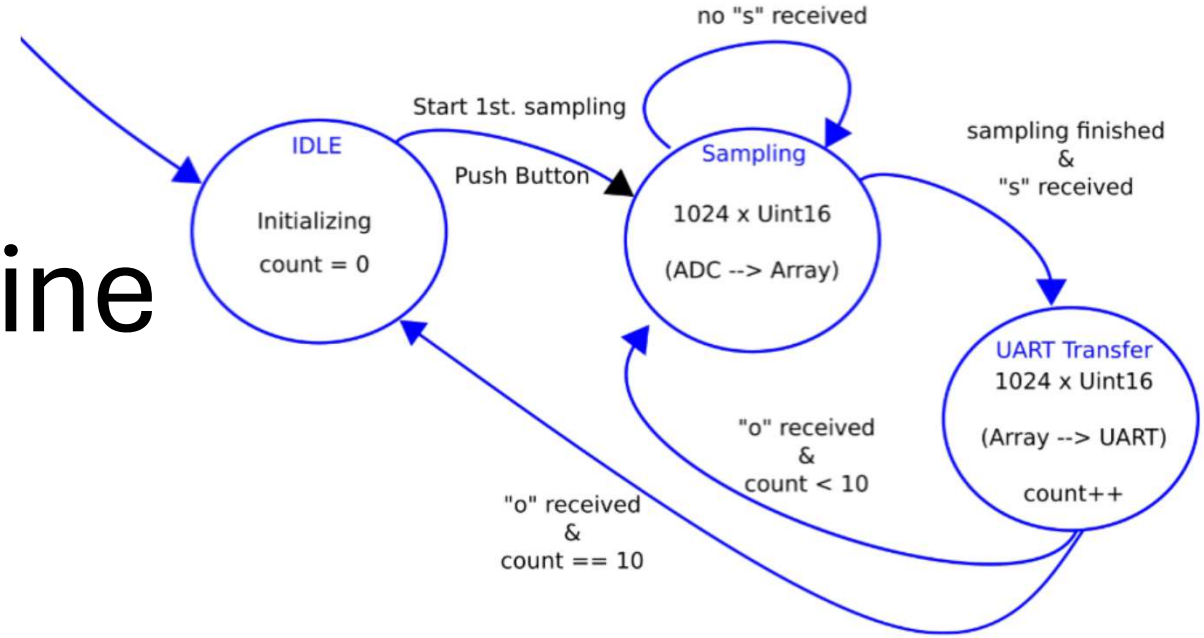
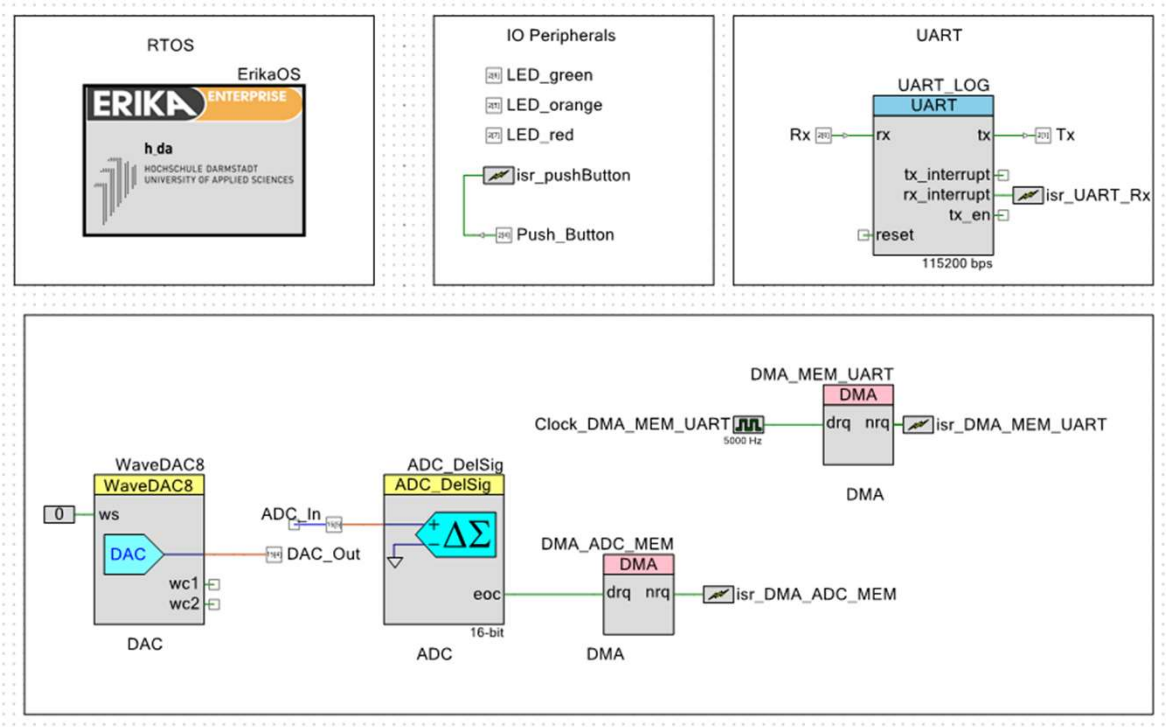
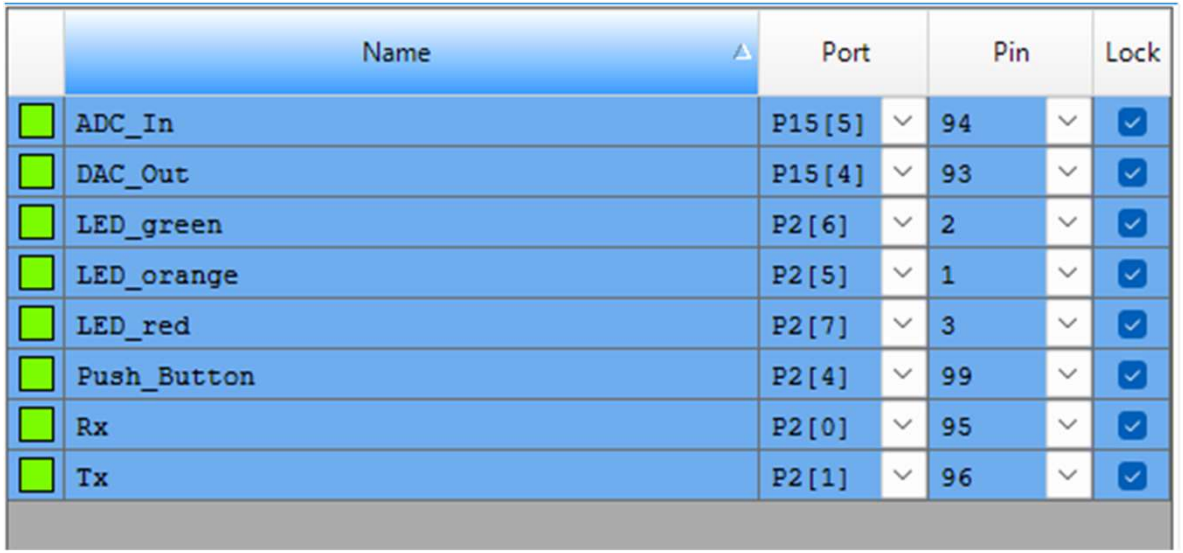
State Machine



Top Design


RTOS
ErikaOS
ERIKA ENTERPRISE
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
IO Peripherals
LED_green
LED_orange
LED_red
isr_pushButton
Push_Button
UART
UART_LOG
UART
Rx
rx
tx
Tx
tx_interrupt
rx_interrupt
tx_en
isr_UART_Rx
reset
115200 bps
DMA_MEM_UART
DMA
Clock_DMA_MEM_UART
5000 Hz
drq nrq
isr_DMA_MEM_UART
DMA
WaveDAC8
WaveDAC8
0
ws
DAC
wc1
wc2
DAC_Out
ADC_In
ADC_DelSig
ADC_DelSig
eoc
16-bit
ADC
DMA_ADC_MEM
DMA
drq nrq
isr_DMA_ADC_MEM
DMA
DAC
DMA

Pin Configuration



	Name	Port	Pin	Lock
	ADC_In	P15[5]	94	☑
	DAC_Out	P15[4]	93	☑
	LED_green	P2[6]	2	☑
	LED_orange	P2[5]	1	☑
	LED_red	P2[7]	3	☑
	Push_Button	P2[4]	99	☑
	Rx	P2[0]	95	☑
	Tx	P2[1]	96	☑

Initializing the OS

```
int main()
{
    CyGlobalIntEnable; /* Enable global interrupts. */

    //Set systick period to 1 ms. Enable the INT and start it.
    EE_systick_set_period(MILLISECONDS_TO_TICKS(1, BCLK__BUS_CLK__HZ));
    EE_systick_enable_int();

    // Start Operating System
    for(;;)
        StartOS(OSDEFAULTAPPMODE);
}
```

Global Object and Macro

```
#define UART_START_VARIABLE 's'
#define UART_FINISH_VARIABLE 'o'

Detector_t movementDetector;
```

Initializing Low Level Drivers and Activating the Required Tasks

```
TASK(tsk_init)
{

    //Init MCAL Drivers
    DETECTOR_init(&movementDetector);

    //Reconfigure ISRs with OS parameters.
    //This line MUST be called after the hardware driver initialisation!
    EE_system_init();

    //Start SysTick
    //Must be done here, because otherwise the isr vector is not overwritten yet
    EE_systick_start();

    //Activate tasks
    ActivateTask(tsk_control);
    ActivateTask(tsk_background);

    TerminateTask();

}
```

DAC and ADC BSW Functions

```c
//DAC States
enum eDAC_ONOFF{
    DAC_OFF = 0,     /**< Turn the LED OFF */
    DAC_ON  = 1      /**< Turn the LED ON */
};
typedef enum eDAC_ONOFF DAC_ONOFF_t;

//ADC States
enum eADC_ONOFF{
    ADC_OFF = 0,     /**< Turn the LED OFF */
    ADC_ON  = 1      /**< Turn the LED ON */
};
typedef enum eADC_ONOFF ADC_ONOFF_t;
```

```c
// Initialize ADC
RC_t ADC_Init();

// Initialize DAC
RC_t DAC_Init();

// Set DAC State
RC_t DAC_Set(DAC_ONOFF_t dacOnOff);

// Set ADC State
RC_t ADC_Set(ADC_ONOFF_t adcOnOff);
```

DMA BSW Functions

```c
// DMA Identifier
enum eDMA_id{
  DMA_ADC_TO_MEMORY,
  DMA_MEMORY_TO_UART
};
typedef enum eDMA_id DMA_id_t;

// DMA States
enum eDMA_ONOFF{
    DMA_OFF = 0,    /**< Turn the LED OFF */
    DMA_ON  = 1     /**< Turn the LED ON */
};
typedef enum eDMA_ONOFF DMA_ONOFF_t;

/* Variable declarations for DMA_ADC_MEM */
#define DMA_ADC_MEM_BYTES_PER_BURST 2
#define DMA_ADC_MEM_REQUEST_PER_BURST 1
#define DMA_ADC_MEM_SRC_BASE (CYDEV_PERIPH_BASE)
#define DMA_ADC_MEM_DST_BASE (CYDEV_SRAM_BASE)

/* Variable declarations for DMA_ADC_MEM */
/* Move these variable declarations to the top of the function */
uint8_t DMA_ADC_MEM_Chan;
uint8_t DMA_ADC_MEM_TD[1];


/* Defines for DMA_MEM_UART */
#define DMA_MEM_UART_BYTES_PER_BURST 2
#define DMA_MEM_UART_REQUEST_PER_BURST 1
#define DMA_MEM_UART_SRC_BASE (CYDEV_SRAM_BASE)
#define DMA_MEM_UART_DST_BASE (CYDEV_PERIPH_BASE)

/* Variable declarations for DMA_MEM_UART */
/* Move these variable declarations to the top of the function */
uint8_t DMA_MEM_UART_Chan;
uint8_t DMA_MEM_UART_TD[1];
```

```c
// Initialize DMA
RC_t DMA_Init();

// SET DMA state
RC_t DMA_Set(DMA_id_t dmaId, DMA_ONOFF_t dmaOnOff);
```

ISRs

```
// Process interrupts received on UART
ISR2(isr_UART_Rx){

    uint8_t data = UART_LOG_GetByte();
    if (data ==  UART_START_VARIABLE){
        SetEvent(tsk_control, ev_sReceived);
    } else if (data == UART_FINISH_VARIABLE){
        SetEvent(tsk_control, ev_oReceived);
    }

}

// Process interrupts when transfer from ADC to memory is completed
ISR2(isr_DMA_ADC_MEM){

    SetEvent(tsk_control, ev_samplingFinished);
}

// Process interrupts when button is pressed
ISR2(isr_pushButton){

    SetEvent(tsk_control, ev_pushButton);
}
```

Structure and States of the Application

```
// States of the statemachine
enum eStates{
    IDLE,
    SAMPLING,
    UART_TRANSFER
};
typedef enum eStates States_t;

// Structure of the Detector object.
struct sDetector{
    States_t detectorState;
    uint8_t  numberOfTransfers;

    boolean_t samplingFinished;
    boolean_t readyToSend;
    boolean_t memoryToUARTFinished;
};
typedef struct sDetector Detector_t;
```

Helper Functions

```
// INIT all low level drivers
RC_t DETECTOR_initDrivers();

// Init Detector object attributes and drivers used by the object
RC_t DETECTOR_init(Detector_t* detector);

// Process events as they get triggered based on a State Machine Architecture
RC_t DETECTOR_processEvents(Detector_t* detector, EventMaskType ev);

// Turns LED ON / OFF based on the detector state
RC_t DETECTOR_setLedState(States_t state);
```

Control task waits for the event and calls the function that processes it

```
TASK(tsk_control)
{
    //Initialize an event and wait for its trigger
    EventMaskType ev = 0;

    while (1){
        WaitEvent(ev_pushButton | ev_reSample | ev_sReceived | ev_send | ev_oReceived);
        GetEvent(tsk_control, &ev);
        ClearEvent(ev);

        // Process the event received
        DETECTOR_processEvents(&movementDetector, ev);

    }

    TerminateTask();
}
```

IDLE State

```
switch (detector -> detectorState){
/*  IDLE states waits for the button to get pushed.
*
*   ev_pushButton:          Triggered when button is pressed. The event turns ON the
*                               ADC, DAC, and DMA modules and sampling is started and data is
*                               transferred from ADC output to a memory location with the help of
*                               DMA module. State changes from IDLE to SAMPLING
*/
case (IDLE):
{
    if (ev & ev_pushButton){

        DMA_Set(DMA_ADC_TO_MEMORY, DMA_ON);
        DAC_Set(DAC_ON);
        ADC_Set(ADC_ON);

        detector -> detectorState = SAMPLING;
        DETECTOR_setLedState(SAMPLING);
    }
}
break;
```

SAMPLING State

```
/*  SAMPLING state waits continously samples the data and waits for a trigger character to arrive on the UART
*   ev_sReceived:           Triggered when UART receives an 's'. Enables a flag to indicate that data
*                               can be sent
*   ev_samplingFinished:    Triggered when ADC finishes sampling. If external system is not ready to receice data,
*                               existing data is discarded and another event is triggered to start resampling
*   ev_reSample:            Triggered when sampling needs to be restarted.
*   ev_send:                Triggered when sampling is finished and external system is ready to recieve data. State changes
*                               from SAMPLING to UART_TRANSFER
*/
case (SAMPLING):
{

    if (ev & ev_sReceived){

        detector -> readyToSend = TRUE;
    }
    if (ev & ev_samplingFinished){

        DMA_Set(DMA_ADC_TO_MEMORY, DMA_OFF);

        if (detector -> readyToSend == FALSE){

            SetEvent(tsk_control, ev_reSample);
        }
        if (detector ->readyToSend == TRUE){

                detector -> readyToSend = FALSE;
                SetEvent(tsk_control, ev_send);
            }
    }
    if (ev & ev_reSample){

        DMA_Set(DMA_ADC_TO_MEMORY, DMA_ON);

    }
    if (ev & ev_send){

        DMA_Set(DMA_ADC_TO_MEMORY, DMA_OFF);
        DMA_Set(DMA_MEMORY_TO_UART, DMA_ON);
        detector -> detectorState = UART_TRANSFER;
        DETECTOR_setLedState(UART_TRANSFER);
    }
}
break;
```

UART_TRANSFER State

```
/*  UART_TRANSFER state transfers the sampled data over UART and waits for a feedback
*   ev_noSReceived:         Trigered when sampling needs to be started again
*   ev_UARTOver:            Triggered when UART receives an 'o'. Marks end of transfer. If number of
*                               transfers < 10. Sampling starts again. If number of transfers = 10, process
*                               is considered complete, and system waits for a button press again. State changes
*                               from UART_TRANSFER to IDLE.
*

*/
case (UART_TRANSFER):
{
    if (ev & ev_oReceived){

        DMA_Set(DMA_MEMORY_TO_UART, DMA_OFF);

        detector -> numberOfTransfers += 1;
        if (detector -> numberOfTransfers < 10){
            detector -> detectorState = SAMPLING;
            DETECTOR_setLedState(SAMPLING);
            SetEvent(tsk_control, ev_reSample);
        }
        else if (detector -> numberOfTransfers == 10){

            detector -> detectorState = IDLE;
            DETECTOR_setLedState(IDLE);
            DAC_Set(DAC_OFF);
            ADC_Set(ADC_OFF);
            detector -> numberOfTransfers = 0;
            detector -> samplingFinished = FALSE;
            detector ->readyToSend = FALSE;
            detector ->memoryToUARTFinished = FALSE;
        }
    }
}
break;
```

MATLAB Script

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Testbench Communication from FreeSoc2 to Matlab
% Version 1.0, Bannwarth, 30.05./332020
%
% Behaviour:
% - Everytime Maltlab writes ‘s’on the UART, the PSoC sends new measurement
%    results and Matalab writes 'o' i5f these data is received.
% - The Script terminates after 10 data transfers.
%
% Using:
% 1. Connect FreeSoc2 to USB (i.e. Power Up)
% 2. Check the correct serial Port Settings
% 3. Start this Matlab Script
% 4. Run this Script
% 5. Press the external Push Button to start measuring
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all;
clear all;
clc;
priorPorts=instrfind;
delete(priorPorts);
PSoC=serial('COM8', 'BaudRate', 115200, 'InputBufferSize', 14000);
fopen(PSoC);
f1 = figure;
count = 1;

flg_data_avai = 0;
fwrite(PSoC,'s','uchar') % means send, I am ready to receive
while(flg_data_avai == 0)
    %fprintf("Transfer in progress: %i, Bytes Available: %d\n", count, PSoC.BytesAvailable); % Print BytesAvailable
       if PSoC.BytesAvailable == 2048
             fwrite(PSoC,'o','uchar') % means I received all expected data
             rx_data_adc = fread(PSoC,1024,'uint16');
             fprintf(" Transfer %i DONE \n",count);

             % Plotting the received data

             figure(f1)
             subplot(2,1,1)
             plot([0:(length(rx_data_adc)-1)],rx_data_adc(1:(length(rx_data_adc))));
             title(['Received Time Domain Data No.:',num2str(count)]);
             subplot(2,1,2)
             plot([0:1023],1/length(rx_data_adc)*20*log10(abs(fft(rx_data_adc))));
             title('FFT -  Matlab');

             % Save the received data
             save(strcat('CW_rx_data_adc_',int2str(count),'.mat'),'rx_data_adc');
             count=count+1;
       end

       if count == 11
           break;
       end

       fwrite(PSoC,'s','uchar') % means send, I am ready to receive
end
fclose(PSoC);
```

MATLAB Output