

Advanced Embedded Systems (TP264)

Projects - Spring 2020

Different applications require well optimized applications running on embedded systems under various constraints, along with requirements for secure and trusted communication. Example application domains include: Remote monitoring of Temperature and Humidity, Real time patient monitoring system, automated smart cities management system, automotive real-time applications, etc.

> The project is an important component of the course. You may use resources you find online. You must give full attribution to your sources (and know what copyright restrictions apply). You must go beyond what you find online
>
> Deliverables: **Report** (Illustrate algorithms, diagrams, screenshots, results, tables, plots; Keep the report short and precise, 2-4 pages), **Final Project Presentation/Demo** (short presentation and demo, together timed for 10 minutes without questions)

Project Ideas/Topics

1.

Cryptographic communication-interaction with an embedded system (Persons: 1)

a. Initial creation of public-private keys using a simple ECDH/ECDSA library (https://github.com/esxgx/easy-ecc)
b. Use symmetric crypto function to send/receive messages to/from an ARM application that displays trusted messages to OLED display
c. Use dedicated procedures (a) and (b) for different applications simultaneously running on the embedded system

(crypto algorithms: https://github.com/B-Con/crypto-algorithms)

2.

Guaranteeing Real-time execution with control of interrupt rate or adapting execution optimizations (Persons: 2)

Investigate different methods to control interrupts (by disabling, postpone, redirect to other core) in order to guarantee timely execution of a computationally intensive task.

Examples of a computationally intensive app: sobel edge detection, Huffman encoding, matrix multiplication, AES encoding/decoding, SHA computation, temperature prediction, sorting, searching.

Alternatively, the application can switch to lower precision (e.g., edge detection, temp prediction, matrix multiplication, compute Huffman dictionary of partial input) but faster execution, switch to parallel execution using more cores (but more power), etc.

3.

Optimal use of DMA data transfers adapting data partitioning depending on data size (Persons: 1)

Investigate the break-even point when utilizing direct-memory-transfers (between external DDR memory and on-chip SRAM memory) of data gives an advantage. Potential computation intensive apps: Huffman encoding, sobel edge detection, matrix multiplication, AES encoding/decoding, SHA computation, temperature prediction, sorting, searching.

4.

Compression optimizations (for performance and for size) using Huffman encoding (Persons: 1)

Use different dictionary for partitions of big data, use memo-ization (previously computed dictionary) to compress next partition of data (e.g. for sequential images).

5.

Demonstration of features using the STM32MP1

Develop examples on STM32MP1 following the next:

https://www.st.com/content/st_com/en/support/learning/stm32-education/stm32-online-training/stm32mp1-online-training.html?ecmp=tt14633_gl_enews_mar2020

6.

Maximum-Likelihood Estimation (design / benchmarking)

Autoregressive–moving-average model

(http://en.wikipedia.org/wiki/Autoregressive%E2%80%93moving-average_model)

```
#define PROB_a 0.001 //0.001
#define PROB_b 0.001 //0.001
float Rt[1000] ;
float getSPRT(float M, float stddev, int current){
float sum, SPRT ;
int j;
sum = 0 ;
for(j=0; j<current; j++) {
```

```
sum += Rt[j] - M/2 ;
}
SPRT = sum * M / stddev ;
return SPRT ;
}
void wlpredict_thread(void)
{
float limit_A, limit_B ;
float stddev ;
float M ;
float avg, sum, SPRT, Tpred ;
u32 timeptr, t0ptr ;
int whole, thousands ;
timeptr = 1 ;
t0ptr = 0 ;
limit_A = log( PROB_b / (1-PROB_a) ) ;
limit_B = log( (1-PROB_b) / PROB_a ) ;
stddev = 3;//2;
M = 3 * sqrt(stddev) ;
Rt[0] = 1;
Rt[1] = 1;
while ( 1 ) {
timeptr++ ;
Rt[timeptr-1] = 1;
SPRT = getSPRT( M, stddev, timeptr ) ;
while ( SPRT<limit_A || SPRT>limit_B) {
u32 j ;
// Update ARMA model:
sum = 0 ;
for(j=t0ptr; j<timeptr; j++) {
sum += Rt[j] ;
}
avg = sum / j ;
stddev = fabs( (avg-3)*(avg-3) ) ; // desired disturbance fixed to 3:
M = 3 * sqrt(stddev) ;
if (SPRT<limit_A) {
Rt[timeptr-1] += fabs( Rt[timeptr-2] - (float) mb_iterations[0]/1000.0 )/4 ;
} else if (SPRT>limit_B) {
Rt[timeptr-1] -= fabs( Rt[timeptr-2] - (float) mb_iterations[0]/1000.0 )/4 ;
}
SPRT = getSPRT( M, stddev, timeptr ) ;
}
//Tpred = (float) local_temp + Rt[timeptr-1] ;
Tpred = (float) mb_iterations[0]/1000.0 + Rt[timeptr-1] ;
}
}
```